VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VIZUALIZACE VÝRAZŮ PROCESNÍ ALGEBRY PI-KALKUL

VISUAL REPRESENTATION OF PI-CALCULUS EXPRESSIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **DAGMAR PROKOPOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/5123/2016/xproko26

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Prokopová Dagmar, Bc.**
Obor: Inteligentní systémy
Téma: **Vizualizace výrazů procesní algebry pi-kalkul**
Visual Representation of Pi-Calculus Expressions
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se obecně s definicí procesní algebry a s různými konkrétními modely a jejich typy relací nad procesy. Zaměřte se na význam výrazů a jeho grafické vyjádření v dostupných publikacích.
2. Blíže se seznamte s procesní algebrou pi-kalkul, s významy jmen, operátorů, redukce a přechodů. Naučte se interpretovat výrazy. Prozkoumejte konkrétní aplikace využívající pi-kalkul a způsoby znázornění procesů a přechodů.
3. Navrhněte grafickou reprezentaci výrazů pi-kalkulu a průběhů redukcí (přechodů). Soustřeďte se zejména na případ předávání netriviálních jmen (procesů). Zvolte také vhodnou textovou reprezentaci (v ASCII).
4. Implementujte aplikaci pro převod textové do grafické reprezentace výrazu pi-kalkulu. Zvažte rozšíření aplikace o převod z grafické na textovou reprezentaci formou grafického editoru.
5. Výsledky zveřejněte jako open-source, zhodnoťte a navrhněte případná rozšíření.

Literatura:

- Robin Milner. Communicating and Mobile Systems: the Pi-Calculus. Springer Verlag, ISBN 0521658691.
- Robin Milner. The Polyadic Pi-Calculus: A Tutorial. In: Logic and Algebra of Specification, vol. 94 of NATO ASI Series, pp. 203-246, Springer Berlin Heidelberg, 1993. ISBN 978-3-642-58041-3. [http://dx.doi.org/10.1007/978-3-642-58041-3_6]
- Davide Sangiorgi, David Walker. The Pi-calculus: A Theory of Mobile Processes. Cambridge University Press, ISBN 0521781779.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rychlý Marek, RNDr., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá problémem vizualizace výrazů procesní algebry pi-kalkul. Teoretická část práce pojednává jak o obecných principech procesních algeber, tak i o specifických vlastnostech konkrétních modelů se zaměřením na procesní algebru π-kalkul. Součástí je rovněž srovnání několika textových a grafických reprezentací výrazů. Hlavní část práce se pak zabývá návrhem a implementací aplikace pro převod textové reprezentace výrazů na grafickou. Vedle textové a grafické reprezentace byla navržena také interní stromová reprezentace určená pro práci s výrazy uvnitř aplikace. V práci jsou také popsány algoritmy pro vyhledání proveditelných redukcí, pro provedení redukce a pro zjednodušení výrazů, které pracují s navrženou stromovou reprezentací.

Abstract

This work deals with the problem of visual representation of Pi-Calculus expressions. The theoretical part of this paper discusses general principles of process algebras as well as specific properties of individual models, with a focus on π-calculus. Also included is the comparison of several text and graphical representations of expressions. The main part of the thesis deals with the design and implementation of an application for converting text representation of expressions into graphical representation. In addition to the text and graphical representation, an internal tree representation designed to work with expressions within the application is also proposed. The thesis also describes algorithms for finding feasible reductions, performing reductions and expression simplification that operate with the proposed tree representation.

Klíčová slova

procesní algebra, pi-kalkul, vizualizace, souběžné systémy, formální specifikace, grafická reprezentace, graf, redukce, výrazy

Keywords

Process algebras, Pi-calculus, Visualization, Concurent systems, Formal specification, Graphical representation, Graph, Reduction, Expressions

Citace

PROKOPOVÁ, Dagmar. *Vizualizace výrazů procesní algebry pi-kalkul.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rychlý Marek.

Vizualizace výrazů procesní algebry pi-kalkul

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D. Rovněž prohlašuji, že jsem v seznamu použité literatury uvedla všechny publikace a zdroje, ze kterých jsem čerpala.

.....
Dagmar Prokopová
24. května 2017

Poděkování

Ráda bych touto cestou poděkovala především vedoucímu mé diplomové práce panu RNDr. Markovi Rychlému, Ph.D. za odborné vedení a poskytované rady i připomínky v průběhu řešení práce. Dále bych chtěla poděkovat svému příteli, který byl vždy připraven konzultovat se mnou nejrůznější problémy, které se vyskytly během vývoje a často přispěl konstruktivními nápady ke zlepšení kvality vytvořené aplikace. V neposlední řadě bych chtěla poděkovat také své rodině za podporu během celého mého studia.

Obsah

Kapitola 1

Úvod

Během druhé poloviny minulého století došlo v oblasti výpočetní techniky k výrazným technologickým pokorokům, co se týče paralelního zpracování. Změny v architektuře počítačů a operačních systémech se brzy začaly projevovat i v oblasti návrhu software, kdy se od jednoduchých sekvenčních aplikací postupně začalo přecházet k rozsáhlejším paralelním a distribuovaným systémům. V dnešní době je již běžnou praxí, že jsou systémy tvořené velkým množstvím komponent (procesů), které spolu vzájemně interagují. Nejsou výjimkou ani případy, kdy se struktura systému a vazby mezi komponentami navíc za běhu dynamicky mění – vznikají nové nebo naopak některé zanikají. O to složitější je pak docílit u takovýchto systémů korektnosti a vyhnout se chybám při návrhu a implemenaci. Klíčovým aspektem pro ověření správnosti návrhu a verifikaci systému je jeho formální specifikace.

Zatímco sekvenční programy bylo dosud možné formálně popisovat pomocí teoretických modelů, kterými jsou například Turingovy stroje či lambda kalkul, s vývojem konkurentních systémů vyvstala potřeba definovat nový formalismus, prostřednictvím něhož by bylo možné specifikovat chování těchto nových systémů a zkoumat jejich vlastnosti – například relaci ekvivalence, možnost uváznutí apod.

Za tímto účelem bylo na přelomu sedmdesátých a osmdesátých let minulého století definováno několik procesních algeber, které na rozdíl od výše uvedených modelů, umožňovali popsat simultální běh více procesů, jejich vzájemnou interakci, komunikaci a synchronizaci. Jednou z nich byla i procesní algebra π-kalkul sloužící jako obecný výpočetní model pro popis mobilních systémů a jejich chování. Mobilia v modelech tohoto kalkulu byla realizována předáváním jmen komunikačních kanálů mezi procesy. π-kalkul se od svého vzniku stal základem několika programovacích jazyků a rovněž bylo navrženo několik jeho grafických reprezentací. Dosud však nebyla vytvořena žádná aplikace, která by umožňovala automatický převod textové reprezentace výrazů na grafickou. Vytvoření takové aplikace bylo cílem diplomové práce.

Tato technická zpráva, jež je výstupem diplomové práce, je věnována jak popisu teoretických základů nezbytných k pochopení řešené problematiky, tak i praktickým aspektům vývoje dané aplikace.

V kapitole 2 nejprve obecně definujeme pojem procesní algebry, popíšeme konkrétní příklady procesních algeber a používaná grafická znázornění. V kapitole 3 se detailněji zaměříme na procesní algebru π-kalkul, specifikujeme syntaxi i sémantiku výrazů této algebry, definujeme relaci redukce a přechodů a pro názornost uvedeme několik příkladů popisu systémů pomocí výrazů π-kalkulu. V kapitole 4 poté stručně popíšeme použití π-kalkulu v praxi a provedeme srovnání existujících nástrojů a grafických reprezentací.

S ohledem na ně poté v kapitole 5 navrhneme vlastní textovou a grafickou reprezentaci výrazů π-kalkulu a specifikujeme požadavky na funkčnost aplikace. Navrhneme rovněž interní reprezentaci výrazů sloužící pro uložení výrazu a jejich zpracování uvnitř aplikace. Na závěr stručně popíšeme jednotlivé funkční celky aplikace a jejich vzájemné propojení.

V kapitole 6 poté zdokumentujeme problémy, které bylo potřeba vyřešit v rámci implementace aplikace, zejména způsoby převodu výrazů mezi navrženými reprezentacemi. Kromě toho uvedeme i algoritmy použité pro provádění redukcí a dalších úprav nad interní reprezentací výrazů.

Závěrem pak zhodnotíme dosažené výsledky a navrhneme možná rozšíření vytvořené aplikace.

Kapitola 2

Procesní algebry

Jak již bylo naznačeno v úvodní kapitole, procesní algebra někdy též označovaná termínem procesní kalkul, je jedním z obecných teoretických modelů použitelných pro formální popis systémů se souběžnými procesy na vysoké úrovni abstrakce [18]. Pojem algebra znamená, že pro popis chování systému a analýzu jeho vlastností jsou používány algebraické metody a axiomatický přístup. I přes velké množství a variabilitu konkrétních modelů procesních algeber, za jejich společný rys můžeme označit fakt, že umožňují popisovat procesy a jejich vzájemné interakce prostřednictvím výrazů složených z relativně malého počtu primitiv a operátorů. Tyto výrazy je přitom možné dále upravovat a analyzovat pomocí kolekce pravidel a axiomů a formálně tak dokazovat některé vlastnosti požadované pro korektní chování systému.

2.1 Základní pojmy

Každou procesní algebru můžeme ve smyslu univerzální algebry definovat jako n-tici (A, $Op_1, Op_2, ...$), kde A značí množinu všech výrazů procesní algebry a Op_i jednotlivé operace nad A definované procesní algebrou. N-ární operaci můžeme matematicky chápat jako zobrazení, které n-tici (zpravidla však dvojici) prvků z A přiřadí prvek z A.

Mezi základní entity většiny procesních algeber patří procesy (agenti), značené zpravidla velkými písmeny latinské abecedy, a jména (události, akce) značené malými písmeny latinské abecedy. Přestože označení i syntax se mohou u různých typů procesních algeber lišit, význam bývá obdobný.

Procesy reprezentují jednotlivé vzájemně komunikující komponenty systému, přičemž každý proces může mít buďto atomickou formu například pokud se jedná o tzv. prázdný proces modelující ukončení výpočtu (označován jako null nebo 0), případně pokud jeho interní reprezentace není předmětem zájmu, a nebo může být tvořen kompozicí několika podprocesů. Formálně je tedy možné proces definovat indukcí.

Jména na rozdíl od procesů nemají jednoznačně interpretovatelný význam, mohou označovat události, proměnné, objekty, zprávy, komunikační kanály či sloužit jako synchronizační prvek. Jsou tak nejen hlavním komunikačním prostředkem, ale i předmětem komunikace mezi procesy.

Pro skládání jednoduchých procesů a jmen do složitějších celků (procesů) slouží **operátory** definované konkrétní procesní algebrou. Prostřednictvím operátorů je zpravidla možné vyjádřit sekvenční, nedeterministickou nebo paralelní kompozici procesů, definovat neko-

nečné rekurzivní chování, omezit platnost jmen jen na určitý kontext nebo vykonat vnitřní či vnější akci.

Akce představuje atomický výpočetní krok, který je proveden systémem za účelem přechodu z jednoho stavu do druhého. Příkladem akce v reálných systémech může být například přijetí či odeslání zprávy. Akce dělíme do následujících dvou typů:

- externí (viditelné) akce - značené malými písmeny latinské abecedy
- interní (tiché) akce - značené zpravidla řeckým písmenem τ

Zatímco interní akce je výsledkem vzájemné interakce dvou paralelně běžících komponent uvnitř popisovaného systému a je tak v zásadě abstrakcí interní reprezentace systému, externí akce vyjadřuje potenciál procesu reagovat s vnějším okolím (například člověkem, který daný systém ovládá). Schopnost procesu vykonat vstupní či výstupní akci se značí prostřednictvím příslušného operátoru zpravidla formou prefixu.

Typy akcí, které zohledňujeme při zkoumání relace ekvivalence dvou procesů mimo jiné určují, zda se jedná o takzvanou silnou nebo slabou ekvivalenci. Slabá ekvivalence abstrahuje od interních akcí, což je žádoucí pokud například chceme zjistit, zda je daný proces nahraditelný jiným, aniž by záměnou došlo ke změně chování z pohledu jeho okolí.

Chování systému můžeme obecně definovat jako množinu všech jeho korektních běhů, kdy během systému rozumíme posloupnost jeho stavů proloženou posloupností odpovídajících akcí. Chování celého systému je vždy dáno kombinací chování všech jeho komponent.

Pro popis operační sémantiky procesních algeber se běžně používá **označený přechodový systém** (LTS), jenž je formálně definován jako trojice $(S, A, \rightarrow)$, kde S označuje množinu všech stavů systému, A množinu akcí (abecedu) a $\rightarrow$ relaci přechodu, pro niž platí $\rightarrow \subseteq S \times A \times S$. Pokud $p, q \in S$ jsou dva stavy systému a $a \in A$ akce, prostřednictvím které systém přejde z výchozího stavu p do cílového stavu q, přechod $(p, a, q) \in \rightarrow$ obvykle značíme $p \xrightarrow{a} q$.

Relace přechodu je ve většině případů definována kolekcí pravidel specifikovaných konkrétní procesní algebrou vycházejících ze sémantiky jednotlivých operátorů a systému axiomů (například axiomy strukturální kongruence).

2.2 Konkrétní příklady

Mezi první a nejvýznamnější zástupce procesních algeber patří algebry CSP, CCS a ACP, které byly nezávisle na sobě definovány různými autory na počátku osmdesátých let 20. století [8]. Tyto algebry se později staly základem pro vznik mnoha dalších procesních algeber, mezi něž můžeme zařadit například i dříve zmíněný π-kalkul.

2.2.1 CSP

Procesní algebra CSP (Communicating Sequential Processes), kterou poprvé představil Tony Hoare roku 1978, byla zpočátku, vzhledem k absenci matematicky definované sémantiky, spíše konkurentním programovacím jazykem, než formálním modelem. Publikace [7] vydaná o několik let později, z níž byly čerpány informace pro tuto kapitolu, však již zahrnovala i formální definici sémantiky.

Jak napovídá název algebry, CSP umožňuje popisovat interakci mezi procesy výhradně pomocí komunikace (tj. výměny zpráv). Pojem "sekvenční" je však zavádějící neboť současná podoba CSP umožňuje definovat procesy rovněž jako paralelní kompozici několika jednodušších podprocesů.

Primitiva

CSP definuje následující dvě třídy primitiv:

- **procesy** reprezentující chování a odpovídající tak procesům popsaných v kapitole 2.1
- **události** ve smyslu jmen popsaných v kapitole 2.1, jakožto prostředek interakce mezi procesy

Abeceda procesu αP je poté množinou všech událostí, kterých se může daný proces P účastnit.

Mezi základní atomické procesy patří prázdný proces $STOP$, který se sám o sobě neúčastní žádných událostí, a tím pádem ani komunikace s ostatními procesy. Reprezentuje uváznutí výpočtu. Dalším atomickým procesem je proces $SKIP$, který prostřednictvím speciální události $\surd$ modeluje úspěšné ukončení výpočtu.

Operátory

CSP poskytuje široké spektrum algebraických operátorů. Prvním z nich je **operátor prefixu** $x \rightarrow P$, který vyjadřuje záměr procesu provést událost x a následně pokračovat jako proces P.

Dalším je **operátor výběru**, který reprezentuje volbu vývoje procesu prostředím. Nechť x, y jsou různé události a P, Q procesy, pak $(x \rightarrow P \mid y \rightarrow Q)$ představuje proces, který nejprve zpracuje událost x respektive y a poté pokračuje jako příslušný proces P respektive Q. Výběr přitom závisí na tom, která z událostí x a y nastane jako první. Výše uvedený proces je možné zapsat obecnější notací $(a : A \rightarrow R(a))$, kde $A = \{x, y\}$ a $R(a)$ odpovídá procesu P v případě, že $a = x$, jinak procesu Q

Pro reprezentaci nedeterminismu definuje algebra dva další operátory. Prvním z nich je **operátor interního výběru** $\sqcap$ a druhým **operátor externího výběru** $\Box$. V případě procesu $P \sqcap Q$ dojde k nedeterministickému výběru některého z procesů P, Q, aniž by prostředí procesu mohlo jakkoliv tuto volbu ovlivnit. Naproti tomu vývoj procesu $P \Box Q$ je možné částečně kontrolovat prostředím. Pokud je totiž akce provedená prostředím první možnou událostí procesu P, ale nikoliv procesu Q, popsaný proces se bude dále vyvíjet jako P a naopak. Tento případ odpovídá definici operátoru výběru $\mid$:

$$((x \rightarrow P) \Box (y \rightarrow Q)) = ((x \rightarrow P) \mid (y \rightarrow Q)) \quad \text{pro} \quad (x \neq y)$$

Pokud je však provedená akce možnou první událostí procesu P i Q, pak je výběr nedeterministický a odpovídá definici operátoru interního výběru $\sqcap$:

$$((x \rightarrow P) \Box (y \rightarrow Q)) = ((x \rightarrow P) \sqcap (y \rightarrow Q)) \quad \text{pro} \quad (x = y)$$

Dalším z operátorů je **paralelní kompozice procesů** $(P \parallel Q)$ vyjadřující konkurentní běh obou procesů. Pokud nastane událost x pro kterou platí, že $x \in (\alpha P \cap \alpha Q)$, dojde ke vzájemné interakci mezi procesy, přičemž tato interakce vyžaduje simultánní realizaci dané události oběma procesy. Ve zvláštních případech, kdy $\alpha P \subseteq \alpha Q$, a tedy každá akce procesu P může nastat pouze tehdy, pokud ji realizuje i proces Q, zatímco proces Q se může účastnit událostí z množiny $(\alpha Q - \alpha P)$ nezávisle na procesu P, mluvíme o vztahu podřízenosti mezi procesy. Pokud je cílem skrýt komunikaci mezi hlavním procesem Q a závislým procesem P, je možné použít asymetrickou notaci $(P /\!/ Q)$.

Na rozdíl od operátoru paralelní kompozice, **operátor prokládání** $(P \,|||\, Q)$ vyjadřuje konkurentní běh procesů bez jakékoliv vzájemné interakce či synchronizace. Každá událost je událostí právě jednoho z procesů, přičemž jsou-li oba procesy schopny vykonat danou událost, nedeterministicky se vybere pouze jeden z nich.

Pro ukrytí vnitřní reprezentace se používá **operátor skrytí**. Proces $(P\backslash C)$, kde C označuje množinu událostí, se chová stejně jako P, až na výskyt všech událostí patřících do této množiny, které jsou daným operátorem skryté. Platí, že $\alpha(P\backslash C) = \alpha P - C$.

Pro vyjádření vztahu mezi sekvenčními procesy P a Q je možné použít několik dalších operátorů, kterými jsou například sekvenční operátor, operátor přerušení či operátor katastrofy. Proces $(P; Q)$ sestrojený pomocí **sekvenčního operátoru** se nejprve chová jako proces P, po jehož úspěšném ukončení se dále chová jako proces Q. Naproti tomu **operátor přerušení** značený symbolem $\triangle$ nevyžaduje úspěšné ukončení procesu. Proces $(P \triangle Q)$ se tak chová nejprve jako proces P, avšak při výskytu první události procesu Q, je vykonávání P přerušeno a dál se pokračuje procesem Q. Zvláštním případem přerušení je výskyt katastrofy. **Katastrofa**, značená symbolem $\lightning$, je taková událost, která se nevyskytuje v abecedě běžícího procesu. Proces $(P \hat{\lightning} Q)$, který je možné zapsat také jako $P \triangle (\lightning \to Q)$, se tedy chová nejprve jako proces P, avšak v případě, že dojde ke katastrofě, jeho běh je přerušen a následně se pokračuje vykonáváním procesu Q, jehož smyslem může být například zotavení se z katastrofy.

Komunikace

Zvláštní třídu událostí představuje komunikace mezi procesy. Komunikace je událost popsaná dvojicí $c.v$, kde c reprezentuje název komunikačního kanálu, na němž se komunikace odehrává a v hodnotu přenášené zprávy. Množina všech zpráv, kterými může proces P komunikovat na kanálu c je definován jako $\alpha c(P) = \{v \mid c.v \in \alpha P\}$. Nechť $v \in \alpha c(P)$, pak $(c!v \to P)$ je takový proces, který nejprve odešle zprávu v na kanálu c a následně pokračuje jako proces P. Obdobně pak $(c?x \to Q(x))$ definuje takový proces, který je připraven přijmout jakoukoliv zprávu x prostřednictvím kanálu c a dále pokračovat jako proces Q parametrizovaný přijatou zprávou.

Paralelní kompozicí těchto dvou procesů dochází k jejich vzájemné komunikaci:

$$(c!v \to P) \parallel (c?x \to Q(x)) = c!v \to (P \parallel Q(v))$$

Procesy které mají pouze jeden vstupní kanál a jeden výstupní kanál jsou označovány jako **roury**. Nechť procesy P a Q jsou rourami se vstupním kanálem in a výstupním out, pak proces $(P \gg Q)$ definuje jejich propojení, přičemž platí, že $\alpha out(P) = \alpha in(Q)$.

Rekurze

Opakující se chování a rekurzi procesů je možné v algebře CSP definovat buď pomocí definiční rovnice ve tvaru X = F(X) nebo konstrukcí $\mu X : A \bullet F(X)$, kde X značí jméno procesu, A abecedu procesu X a F(X) výraz, v němž se vyskytuje reference na definovaný proces uvozená prefixem. Rekurze umožňuje definovat jeden proces jako řešení jedné definiční rovnice. V případě nepřímé rekurze je nutné použít soustavu definičních rovnic. Použitím indexace proměnných je možné specifikovat nekonečnou soustavu rovnic. Příkladem něchť je specifikace chování objektu P, který se může vykonáním akce up pohybovat směrem nahoru a akcí $down$ dolů:

$$P_{n+1} = (up \to P_{n+2}) \mid (down \to P_n)$$

Značení

Pro modelování většího počtu procesů se stejným chováním avšak odlišnou abecedou, umožňuje algebra CSP využít pro specifikaci jednotlivých procesů a událostí značení. Označený proces $label : P$ se pak účastní události $label.x$ ve všech takových případech, kdy by se proces P účastnil události x. Toto je možné zobecnit zápisem $L : P$, kde L je množina všech značení a výběr konkrétního značení závisí na prostředí.

Stopa (Trace)

Jedním z hlavních denotačních modelů procesní algebry CSP je model stop. Stopa je konečná posloupnost událostí, kterých se proces účastnil během svého běhu. Prázdná stopa se značí $\langle\rangle$, stopa s jednou událostí $\langle x\rangle$, se dvěma $\langle x, y\rangle$ atd. CSP definuje nad stopami poměrně velké množství operací, mezi něž patří například spojení stop, omezení jen na některé symboly, výběr prvního prvku, reverzace, prokládání a další. Popis těchto operací je však již nad rámec této práce.

2.2.2 CCS

Procesní algebra CCS (Calculus of Communicationg Systems), jejímž autorem je Robin Milner, byla poprvé oficiálně představena v publikaci [10] z roku 1980. Tato algebra je založena na myšlence pozorování chování systémů, tvořených komunikujícími agenty, z pohledu vnějšího pozorovatele. Autor se ve své práci kromě specifikace syntaxe a sémantiky samotné algebry věnuje ve značné míře rovněž popisu různých typů relací (ekvivalence a kongruence) nad procesy a jejich vztahy.

Primitiva

Hlavními prvky procesní algebry CCS jsou **agenti**, jakožto nezávislé komponenty modelovaného systému schopné synchronní komunikace, a **značení** (labels). Značení se dle typu portu, který popisují, dále dělí na jména (označující vstupní porty) a ko-jména (označující výstupní porty).

Množina všech značení $\bigwedge$ je dána sjednocením množiny jmen $\triangle$ s množinou všech kojmen $\bar{\triangle}$. Množiny $\triangle$ a $\bar{\triangle}$ jsou vzájemně disjunktní a je nad nimi definována relace bijekce:

$$a(\in \triangle) \rightarrow \bar{a}(\in \bar{\triangle})$$

Mezi základní entity jazyka procesní algebry CCS řadíme **hodnotové výrazy**, tvořené proměnnými (značíme obvykle x, y, ...), symboly konstant a funkcí, dále pak **identifikátory chování** o určité aritě ,udávající počet hodnotových parametrů, a **výrazy chování** specifikující význam jednotlivých identifikátorů.

Operátory

Procesní algebra CCS definuje celkem 6 základních operací, které dělí na statické a dynamické.

Mezi dynamické operace patří nulární operace produkující hodnotu prázdného procesu NIL, binární operace **sčítání** (značená symbolem $+$) a unární operace komunikační **akce** (značená prefixem $\lambda.$ pro každé $\lambda \in \bigwedge \cup \{\tau\}$). Mezi dynamické operace poté řadíme operaci **paralelní kompozice** (značená symbolem $|$), operaci **restrikce** (značená pomocí postfixu

$\backslash\delta$, kde $\delta \in \triangle$) a operaci **přeznačení** (značená postfixem $[f]$, kde f je zobrazení z L do M přičemž $L, M \subseteq \bigwedge$). Nechť α_1 až α_n jsou vzájemně rozdílná jména z množiny M a β_1 až β_n odpovídající navzájem rozdílná jména z množiny L, pak přeznačení zpravidla zapisujeme ve tvaru $[\alpha_1/\beta_1, ..., \alpha_n/\beta_n]$.

Nechť B, B_1 a B_2 označuje výrazy chování a b identifikátor chování arity n parametrizovaný hodnotovými výrazy E_1 až E_n. Syntax výrazu chování je poté dána následující gramatikou:

$$B ::= b(E_1, ..., E_n) \mid NIL \mid B_1 + B_2 \mid \lambda.B_1 \mid B_1|B_2 \mid B_1\backslash\delta \mid B_1[f]$$

Pořadí vyhodnocování výrazů je stanoveno následujícím uspořádáním dle priority operátorů, přičemž pro vynucení jiného pořadí je nutné použít ve výrazech závorky:

$$\{\text{Restrikce}, \text{Přeznačení}\} > \text{Akce} > \text{Kompozice} > \text{Sčítání}$$

Pomocí základních operátorů umožňuje CCS definovat odvozené operátory, například operátor **zřetězení** $\frown$. Nechť x a y označuje komponenty s jedním vstupním a jedním výstupním portem. Vsupní port komponenty y označíme α, přičemž platí, že $\alpha \in M$, kde M je množina všech značení komponenty y. Výstupní port komponenty x označíme β, přičemž obdobně platí, že $\beta \in L$, kde L je množina značení komponenty x. Pak $x \frown y = (x[\delta/\beta] \mid y[\delta/\alpha])\backslash\delta$, přičemž $\delta \notin (L \cup M)$.

Rekurze

Program v jazyce procesní algebry CCS je tvořen kolekcí definičních rovnic v následujícím tvaru:

$$b(x_1, ..., x_n) \stackrel{\text{def}}{=} B_b,$$

kde $b(x_1, ..., x_n)$ je identifikátor chování a B_b výraz chování. Jedna definiční rovnice tak odpovídá specifikaci jednoho procesu. Rekurzi (i nepřímou) je možné modelovat tak, že výraz chování obsahuje referenci na konkrétní identifikátor chování, tedy například

$$b(x_1, ..., x_n) \stackrel{\text{def}}{=} \tau.b(E_1, ..., E_n)$$

Toto volání se pak vyhodnotí jako

$$b(x_1, ..., x_n) \stackrel{\text{def}}{=} \tau.B_b\{E_1/x_1, ..., E_n/x_n\} \ ,$$

kde $B_b\{E_1/x_1, ..., E_n/x_n\}$ nebo významově ekvivalentní zápis $B_b\{\tilde{E}/\tilde{x}\}$, označuje substituci hodnotového výrazu E_i za všechny volné výskyty proměnné x_i ve výrazu chování B_b pro $(1 \leq i \leq n)$.

Komunikace

K obdobné substituci dochází při předávání hodnot prostřednictvím komunikačních kanálů. CCS totiž umožňuje nejen synchronizaci procesů pomocí jmen ve formě prefixu, ale také komunikaci předáváním proměnných svázaných s daným jménem. Nechť $p \stackrel{\text{def}}{=} \alpha x.B$ specifikuje proces, který přijme na vstupním portu α hodnotu, kterou naváže na proměnnou x a následně pokračuje dle výrazu chování B. Definujme obdobný proces $q \stackrel{\text{def}}{=} \bar{\alpha}E.NIL$, který pouze na výstupním portu $\bar{\alpha}$ odešle hodnotový výraz E. Jsou-li tyto procesy uspořádány do paralelní kompozice $p \mid q$, pak dojde k jejich vzájemné interakci na komunikačním kanálu α, což zapíšeme následovně: $p \mid q \xrightarrow{\tau} B\{E/x\} \mid NIL$. Všechny volné výskyty proměnné x ve výrazu chování B jsou přitom nahrazeny přijatým hodnotovým výrazem E.

Relace přechodu

Nad množinou všech výrazů chování definuje algebra CCS binární relaci $\xrightarrow{\mu v}$ pro každé $\mu \in \bigwedge \cup \{\tau\}$ a příslušnou hodnotu v. $B \xrightarrow{\mu v} B'$ je možné chápat ve významu provedení atomické akce agentem, která má za následek přechod z jednoho stavu do druhého (neboli redukci výrazu chování B na výraz B'). Tuto relaci je možné definovat indukcí na struktuře výrazů chování, neboť všechny atomické akce složeného výrazu mohou být odvozeny z atomických akcí jeho komponent.

Nechť $s = \mu_1 v_1 ... \mu_n v_n \in (\bigwedge \cup \{\tau\})^*$ je posloupnost akcí. Pak můžeme psát $B \xrightarrow{s} B'$ ve významu **derivace** $B = B_0 \xrightarrow{\mu_1 v_1} B_1 \xrightarrow{\mu_2 v_2} B_2 ... \xrightarrow{\mu_n v_n} B_n = B'$ pro nějaké B_0 až B_n, kde $(n \geq 0)$. Úplná derivace je buď nekonečná, nebo taková, která již dále nemůže být rozšiřována (tj. $B_n = NIL$). Množina všech derivací výrazu chování, který specifikuje modelovaný proces, podává kompletní informaci o možnostech vývoje daného procesu uvnitř systému.

Relace přechodu je definována prostřednictvím množiny **axiomů** a **odvozovacích pravidel** popisujících sémantiku výše zmíněných operací nad výrazy procesní algebry CCS. V následujícím výčtu základních konstrukcí výrazů chování, budou odvozovací pravidla uváděna vždy ve tvaru

$$\frac{\text{předpoklad}}{\text{závěr}}$$

NIL

NIL je označení pro prázdný výraz, který není schopen provádět žádné akce. Z toho důvodu mu nepřiřazujeme žádné odvozovací pravidlo, ale uvádíme jej pouze pro úplnost.

Identifikátor

Předpokládejme, že identifikátor chování b je definován klauzulí $b(x_1, ...x_n) \stackrel{\text{def}}{=} B_b$, přičemž platí, že množina všech volných proměnných v B_b je podmnožinou $\{x_1, ..., x_n\}$. Pravidlo IDE říká, že každý parametrizovaný identifikátor chování může provádět stejné akce jako odpovídající instance výrazu na pravé straně jeho definiční rovnice.

$$IDE: \qquad \frac{B_b\{v_1/x_1, ..., v_n/x_n\} \xrightarrow{\mu v} B'}{b(v_1, ..., v_n) \xrightarrow{\mu v} B'}$$

Sčítání

Operace sčítání má význam nedeterministického výběru jednoho z výrazů chování. Platí, že atomické akce součtu odpovídají atomickým akcím sčítaných výrazů.

$$SUM(1): \qquad \frac{B_1 \xrightarrow{\mu v} B_1'}{B_1 + B_2 \xrightarrow{\mu v} B_1'}$$

$$SUM(2): \qquad \frac{B_1 \xrightarrow{\mu v} B_2'}{B_1 + B_2 \xrightarrow{\mu v} B_2'}$$

Komunikační akce

CCS definuje 3 axiomy komunikačních akcí. $\alpha \in \triangle$ označuje vstupní akci, $\bar{\alpha} \in \bar{\triangle}$ výstupní akci a τ interní skrytou akci.

ACT (1): $\alpha x_1, ..., x_n.B \xrightarrow{\alpha(v_1,..,v_n)} B\{v_1/x_1, ..., v_n/x_n\}$

ACT (2): $\bar{\alpha} v_1, ..., v_n.B \xrightarrow{\bar{\alpha}(v_1,...,v_n)} B$

ACT (3): $\tau.B \xrightarrow{\tau} B$

Paralelní kompozice

Pravidla COM(1) a COM(2) paralelní kompozice popisují situaci, v níž jeden z agentů tvořících kompozici provádí akci, které se však druhý nijak neúčastní. COM(3) poté vyjadřuje vzájemnou komunikaci mezi komponentami, která vede k interní akci.

$$COM(1): \quad \frac{B_1 \xrightarrow{\mu v} B_1'}{B_1 \mid B_2 \xrightarrow{\mu v} B_1' \mid B_2}$$

$$COM(2): \quad \frac{B_2 \xrightarrow{\mu v} B_2'}{B_1 \mid B_2 \xrightarrow{\mu v} B_1 \mid B_2'}$$

$$COM(3): \quad \frac{B_1 \xrightarrow{\lambda v} B_1' \quad B_2 \xrightarrow{\bar{\lambda} v} B_2'}{B_1 \mid B_2 \xrightarrow{\tau} B_1' \mid B_2'}$$

Restrikce

Smyslem podmínky $\mu \notin \{\alpha, \bar{\alpha}\}$ v pravidle RES je zajistit, že $B\backslash\alpha$ nemůže provést akci αv ani $\bar{\alpha} v$.

$$RES: \quad \frac{B \xrightarrow{\mu v} B'}{B\backslash\alpha \xrightarrow{\mu v} B'\backslash\alpha} \quad , \quad \mu \notin \{\alpha, \bar{\alpha}\}$$

Přeznačení

Pro funkci přeznačení platí $f(\tau) = \tau$.

$$REL: \quad \frac{B \xrightarrow{\mu v} B'}{B[f] \xrightarrow{(f\mu)v} B'[f]}$$

Odvozovací pravidla uvedená výše odpovídají pravidlům uvedeným v [10] V jiných publikacích zabývajících se procesní algebrou CCS je možné nalézt pravidla mírně odlišná.

2.2.3 ACP

Poslední ze základní trojice procesních algeber je ACP (Algebra of Communicating Processes) publikovaná dvojicí autorů Jan Bergstra a Jan Willem Klop. Tato procesní algebra byla inspirována algebrou CCS, avšak více než v případě jejích předchůdců, byl při jejím vývoji využit ryze algebraický přístup s cílem vytvořit obecný axiomatický systém pro popis procesů. Výsledky byly popsány v článku [2], v němž se vůbec poprvé objevil pojem procesní algebra. Informace v této kapitole byly čerpány z [3].

Primitiva

ACP poskytuje způsob popisu procesů ve tvaru algebraických výrazů tvořených kompozicí dalších procesů a primitiv. Základními primitivy algebry jsou **atomické akce** značené malými písmeny latinské abecedy. Některé akce se speciálním významem se značí malými písmeny řecké abecedy. Jedná se o akci δ reprezentující uváznutí a tichou interní akci τ.

Pro specifikaci nekonečných procesů, zavádí algebra **rekurzivní proměnné**, značené velkými písmeny latinské abecedy. Proces vykonávající nekonečný výpočet prováděním akce a je tak možné specifikovat rekurzivní rovnicí ve tvaru $X = aX$.

Operátory

Atomické akce jsou formovány do výrazů prostřednictvím **operátorů**. ACP definuje významově obdobné operace jako algebry CSP a CCS, přičemž jejich vlastnosti jsou vždy formálně popsány množinou axiomů.

Hlavními konstruktory výrazů základní procesní algebry jsou operátor **sekvenční kompozice** (značený symbolem $\cdot$), jenž vyjadřuje postupné řazení akcí, a operátor **alternativní kompozice** (značený symbolem $+$) vyjadřující výběr jedné z akcí. Proces posaný výrazem $(a + b) \cdot c$ tedy nejprve provede buď akci a, nebo akci b a následně pokračuje vykonáním akce c.

Axiomy 2.1 definují základní vlastnosti těchto operací, kterými jsou asociativita obou zmíněných operací, komutativita a idempotence operace $+$ a distributivita operace $\cdot$ vůči operaci $+$.

$$\begin{aligned} (x + y) + z &= x + (y + z) \\ (x \cdot y) \cdot z &= x \cdot (y \cdot z) \\ x + y &= y + x \\ x + x &= x \\ (x + y) \cdot z &= x \cdot z + y \cdot z \end{aligned} \tag{2.1}$$

Axiomy 2.2 definují vztah těchto operací k akci uváznutí δ.

$$\begin{aligned} \delta + x &= x \\ \delta \cdot x &= \delta \end{aligned} \tag{2.2}$$

Pro vyjádření **paralelní kompozice** procesů definuje algebra operátor sloučení (značený symbolem $\|$) a pomocný operátor levého sloučení (značený $\lfloor\!\lfloor$). Nechť x a y jsou procesy, pak jejich paralelní kompozice $(x \parallel y)$ označuje takový proces, který nejprve provede výpočetní krok nad jedním z procesů x nebo y a následně pokračuje jako paralelní kompozice jejich zbývajících částí. Operátor levého sloučení má obdobný význam, avšak s omezením specifikujícím, že počáteční akce musí být akcí levého operandu. Zatímco tak proces popsaný výrazem $(a \cdot b) \parallel (c \cdot d)$ může provádět akce a, b, c, d v pořadí daném kteroukoliv posloupností z množiny $\{abcd, acbd, acdb, cabd, cadb, cdab\}$, proces popsaný výrazem $(a \cdot b) \lfloor\!\lfloor (c \cdot d)$ musí vždy provést nejprve akci a a teprve poté pokračovat standardním prokládáním zbývajících akcí. Možné posloupnosti provádění akcí jsou tak pouze $\{abcd, acbd, acdb\}$.

Pro popis **interakcí** mezi procesy uspořádanými do paralelní kompozice slouží komunikační funkce (značená symbolem $|$). Tato funkce splňující axiomy 2.3 je definovaná jako zobrazení dvojice atomických akcí na jinou atomickou akci.

$$
\begin{aligned}
a \mid b &= b \mid a \\
(a \mid b) \mid c &= a \mid (b \mid c) \\
\delta \mid a &= \delta
\end{aligned}
\tag{2.3}
$$

Úpravou formátu atomických akcí a komunikační funkce umožňuje ACP modelovat i předávání datových hodnot mezi procesy. Nechť D je množina datových hodnot d, P množina portů p a nechť množina atomických akcí procesní algebry je rozšířena o následující akce.

- vstupní akce $rp(d)$ - reprezentující čtení hodnoty d z portu p
- výstupní akce $wp(d)$ - reprezentující zápis hodnoty d na port p
- transakce $cp(d)$ - reprezentující komunikaci hodnoty d skrz port p

Pro každé dvě odpovídající si vstupní a výstupní akce poté platí: $wp(d) \mid rp(d) = cp(d)$. Dvojice akcí nekompatibilní pro vzájemnou komunikaci, jsou pak obvykle redukovány na akci δ.

Pomocí operátoru komunikačního sloučení (značený symbolem $|$), který je rozšířením komunikační funkce na výrazy (nejen atomické akce), je možné definovat plnohodnotný operátor **sloučení s komunikací** (značený symbolem $\parallel$) [1]. Vlastnosti tohoto operátoru a pomocných operátorů levého sloučení a komunikačního sloučení jsou popsány skupinou axiomů 2.4.

$$
\begin{aligned}
x \parallel y &= x \between y + y \between x + x \mid y \\
(a \cdot x) \between y &= a \cdot (x \parallel y) \\
a \between y &= a \cdot y \\
(x + y) \between z &= x \between z + y \between z \\
(a \cdot x) \mid b &= (a \mid b) \cdot x \\
a \mid (b \cdot x) &= (a \mid b) \cdot x \\
(a \cdot x) \mid (b \cdot y) &= (a \mid b) \cdot (x \parallel y) \\
(x + y) \mid z &= x \mid z + y \mid z \\
x \mid (y + z) &= x \mid y + x \mid z
\end{aligned}
\tag{2.4}
$$

Pro odstranění neúspěšných pokusů o komunikaci slouží operátor **zapouzdření** značený ve formě prefixu ∂_H, kde H je podmnožinou množiny všech atomických akcí. Tento operátor nahradí všechny atomické akce $a \in H$ akcí δ. Tím pádem nebude docházet prostřednictvím těchto akcí ke komunikaci zapouzdřeného systému s prostředím. Význam operátoru je popsán množinou axiomů 2.5.

$$
\begin{aligned}
\partial_H(a) &= a \quad if a \notin H \\
\partial_H(a) &= \delta \quad if a \in H \\
\partial_H(x + y) &= \partial_H(x) + \partial_H(y) \\
\partial_H(x \cdot y) &= \partial_H(x) \cdot \partial_H(y)
\end{aligned}
\tag{2.5}
$$

[1] Dříve definovaný operátor **sloučení** je instancí operátoru **sloučení s komunikací**, kdy komunikační funkce je definovaná předpisem $a \mid b = \delta$ pro všechny dvojice atomickýh akcí a a b.

Posledním operátorem je operátor **abstrakce** značený τ_I, kde I je podmnožinou množiny všech atomických akcí. Tento operátor umožňuje nahradit všechny akce $a \in I$ tichou akcí τ a tím abstrahovat interní reprezentaci modelovaného systému. Sémantika operátoru je popsána množinou axiomů 2.6.

$$\begin{aligned} \tau_I(\tau) &= \tau \\ \tau_I(a) &= a \quad if a \notin I \\ \tau_I(a) &= \tau \quad if a \in I \\ \tau_I(x+y) &= \tau_I(x) + \tau_I(y) \\ \tau_I(x \cdot y) &= \tau_I(x) \cdot \tau_I(y) \end{aligned} \tag{2.6}$$

V některých případech je možné interní tichou akci τ z výrazů odstranit. Axiomy 2.7 popisují vlastnosti tiché akce a možnosti jejího bezpečného odstranění.

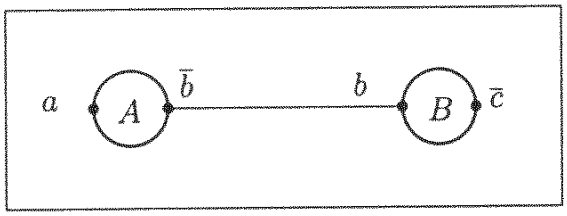
$$\begin{aligned} x \cdot \tau &= x \\ \tau \cdot x &= \tau \cdot x + x \\ a \cdot (\tau \cdot x + y) &= a \cdot (\tau \cdot x + y) + a \cdot x \end{aligned} \tag{2.7}$$

2.3 Grafické znázornění

Ve většině publikací věnujících se problematice procesních algeber je použit jistý způsob grafického znázornění jak pro popis struktury modelovaného systému, tak i pro popis jeho chování. Příklady grafických znázornění v této podkapitole jsou převzaty z několika publikací, konkrétně [13], [20] a [7].

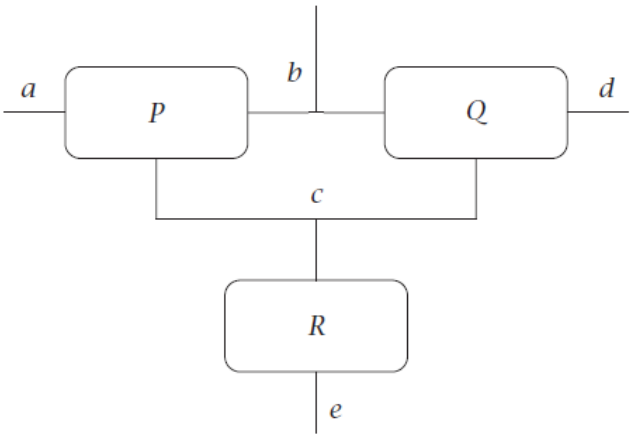
2.3.1 Strukturální pohled

Struktura modelovaného systému je zpravidla vizualizována pomocí **diagramu propojení**. Jednotlivé uzly (případně bloky) reprezentují konkrétní komponenty systému (procesy) a hrany vazby mezi nimi (komunikační kanály). Hrany jsou v místě napojení na procesy označeny symboly, které reprezentují vstupní či výstupní porty daného procesu. Každé dva porty vyskytující se na jedné hraně jsou vždy vzájemně komplementární, tj. jeden je vstupní a druhý výstupní. Společně tak reprezentují vstupně-výstupní akci jakožto prostředek interakce propojených komponent.



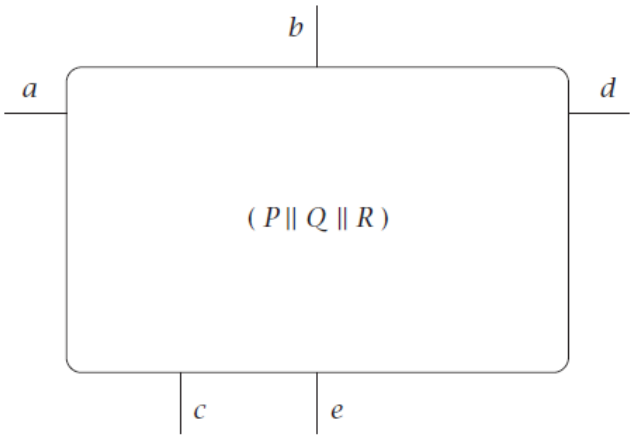
Obrázek 2.1: Grafické znázornění struktury systému tvořeného dvěma komponentami A a B s označenými vstupními a výstupními porty — převzato z [13].

Místo označení jednotlivých portů je možné ohodnotit příslušnou hranu přímo symbolem vstupně-výstupní akce. V takovém případě má smysl použít orientovaný graf, kdy každá hrana udává směr od výstupního portu k vstupnímu.

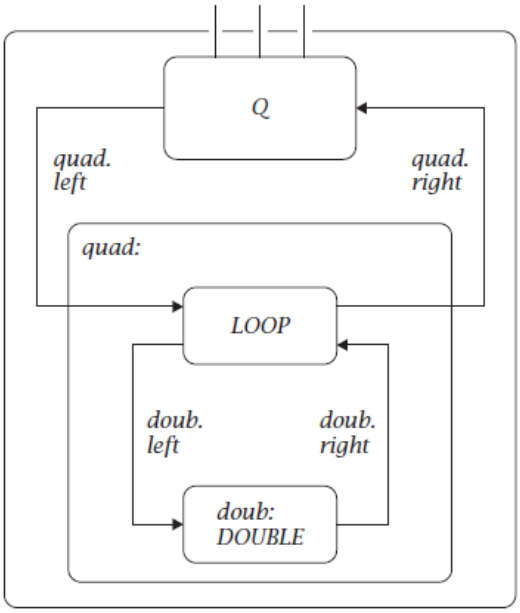


Obrázek 2.2: Grafické znázornění struktury systému tvořeného třemi komponentami P, Q a R s označením hran (nikoliv portů) — převzato z [7].

Procesy znázorněné diagramem propojení jsou obvykle modelovány ve smyslu černé skříňky s rozhraním. Jelikož však každý proces může být tvořen kompozicí dalších procesů, závisí vždy na tvůrci diagramu, jakou míru abstrakce pro modelovaný systém zvolí. Na obrázku 2.3 je znázorněn jediný proces tvořený paralelní kompozicí procesů P, Q a R znázorněných na obrázku 2.2 Na obrázku 2.4 je pak možné vidět komponentu *quad* zakreslenou v diagramu včetně vnitřní reprezentace.

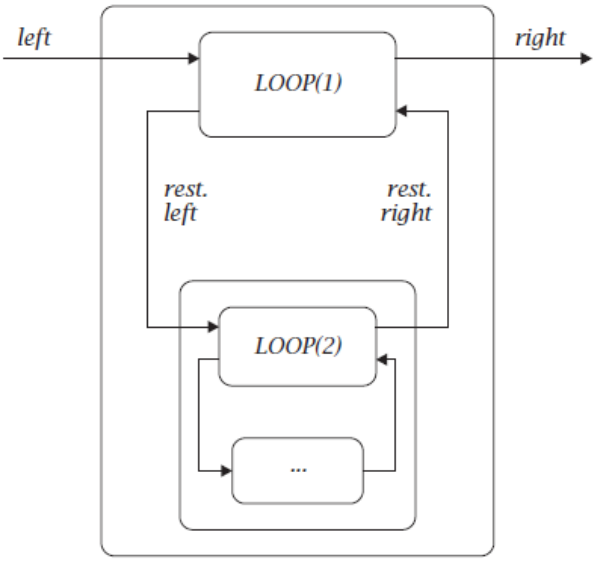


Obrázek 2.3: Grafické znázornění struktury systému tvořeného jedinou komponentou, která je paralelní kompozicí výše uvedených komponent P, Q a R. Vzájemná komunikace mezi těmito komponentami je abstrahována — převzato z [7].

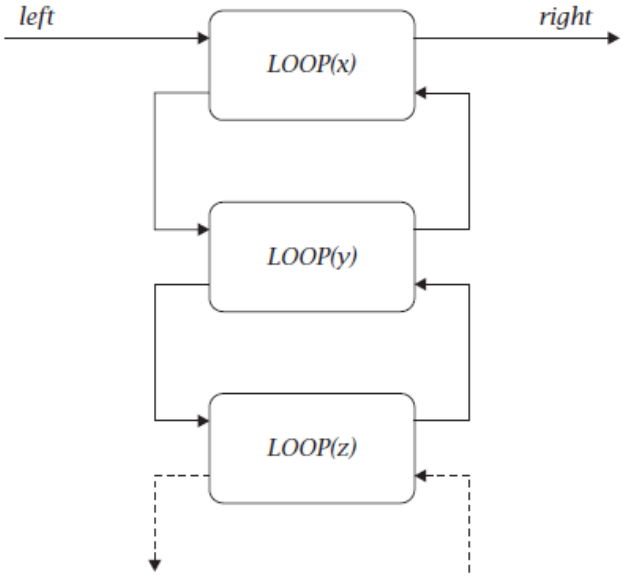


Obrázek 2.4: Grafické znázornění struktury systému s komponentami Q a *quad*. Interní reprezentace komponenty *quad* je explicitně znázorněna — převzato z [7].

Procesní algebry jakožto teoretické modely umožňují prostřednictvím příslušných operátorů modelovat například i nekonečné zřetězení nebo zanoření procesů. Takovéto případy přirozeně není možné popsat diagramem v plném rozsahu, proto je vždy zobrazena pouze část struktury systému a opakující se procesy jsou znázorněny formou teček. Příklad takového zjednodušení je možné vidět na obrázcích 2.5 a 2.6.

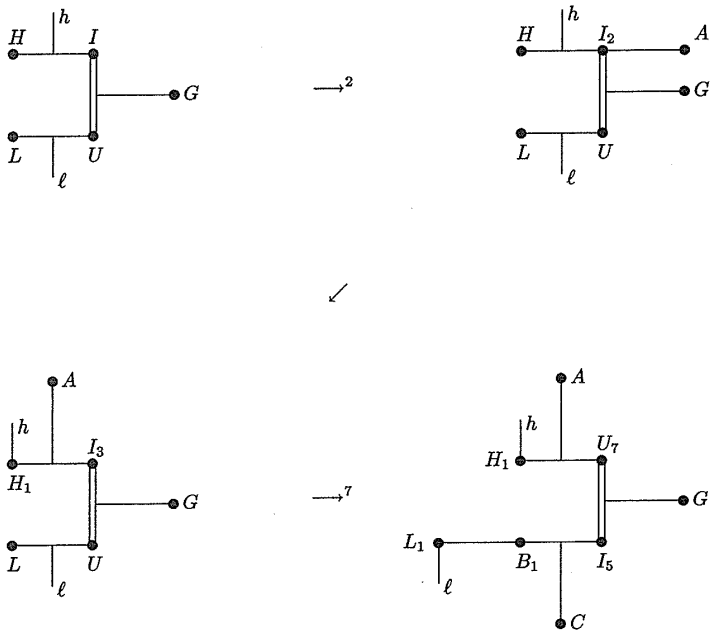


Obrázek 2.5: Grafické znázornění struktury systému s nekonečným zanořením komponent — převzato z [7].



Obrázek 2.6: Grafické znázornění lineární struktury systému z obr. 2.5 — převzato z [7].

Prostřednictvím diagramu propojení je možné modelovat pouze statickou strukturu procesu. Vývoj systému s dynamickou architekturou, během něhož může docházet ke změně struktury systému (např. vzniku nových komponent) je však možné znázornit sekvencí těchto diagramů. Obrázek 2.7 zobrazuje několik výpočetních kroků procesu.

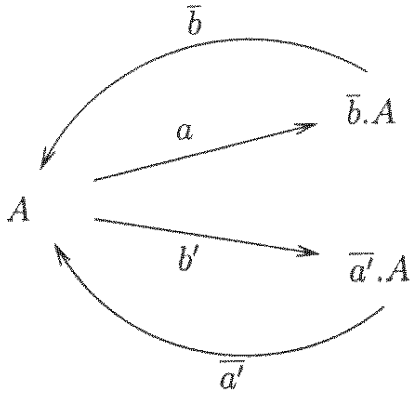


Obrázek 2.7: Grafické znázornění části vývoje systému pomocí diagramů propojení — převzato z [20].

2.3.2 Behaviorální pohled

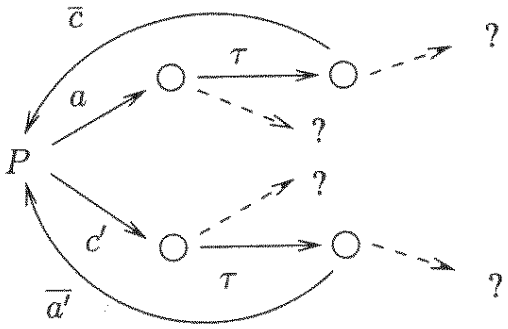
Pro popis chování a dynamických vlastností systému se zpravidla používá **přechodový diagram**, který odpovídá příslušnému označenému přechodovému systému (viz 2.1). Přechodový systém, podobně jako konečný automat, je možné znázornit pomocí orientovaného grafu, jehož jednotlivé uzly představují stavy systému a značené orientované hrany validní přechody mezi nimi.

Na obrázku 2.8 je znázorněn přechodový diagram procesu A specifikovaného v CCS pomocí definiční rovnice $A \stackrel{\text{def}}{=} a.\overline{b}.A + b'.\overline{a'}.A$



Obrázek 2.8: Přechodový diagram znázorňující možný vývoj procesu A — převzato z [13]

Přechodové diagramy je možné použít i pro analýzu chování konkurentních procesů. Na obrázku 2.9 je znázorněna část procesu $P \stackrel{\text{def}}{=} new\, bb'\,(A|B)$, jenž je paralelní kompozicí výše uvedeného procesu A a procesu B definovaného rovnicí $B \stackrel{\text{def}}{=} A\langle b, b', c, c'\rangle$.

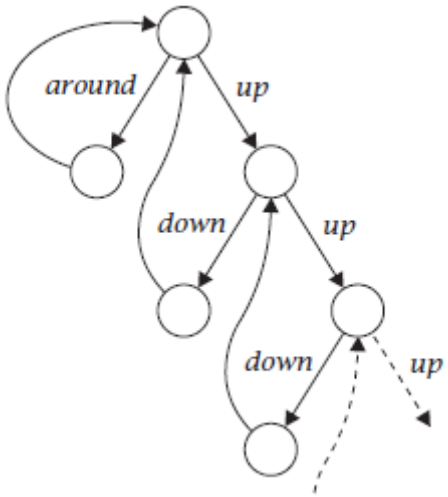


Obrázek 2.9: Část přechodového diagramu znázorňující možný vývoj procesu P — převzato z [13]

Přechod z jednoho stavu do druhého provedením interní reakce je v tomto případě označen symbolem τ. Někdy se však takový přechod ponechává bez označení.

Na rozdíl od konečných automatů, množina stavů ani množina přechodů přechodového systému nemusí být konečná ani spočetná, což představuje jistý problém pro grafické zob-

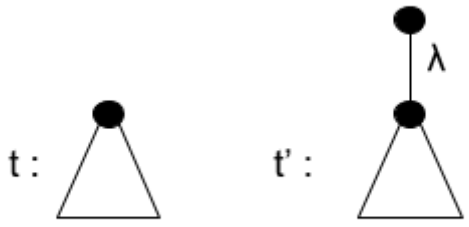
razení takového systému a přechodový graf tím pádem nemusí být kompletní (viz obrázek 2.10).



Obrázek 2.10: Část přechodového diagramu znázorňující přechodový systém s nekonečným počtem stavů — převzato z [7]

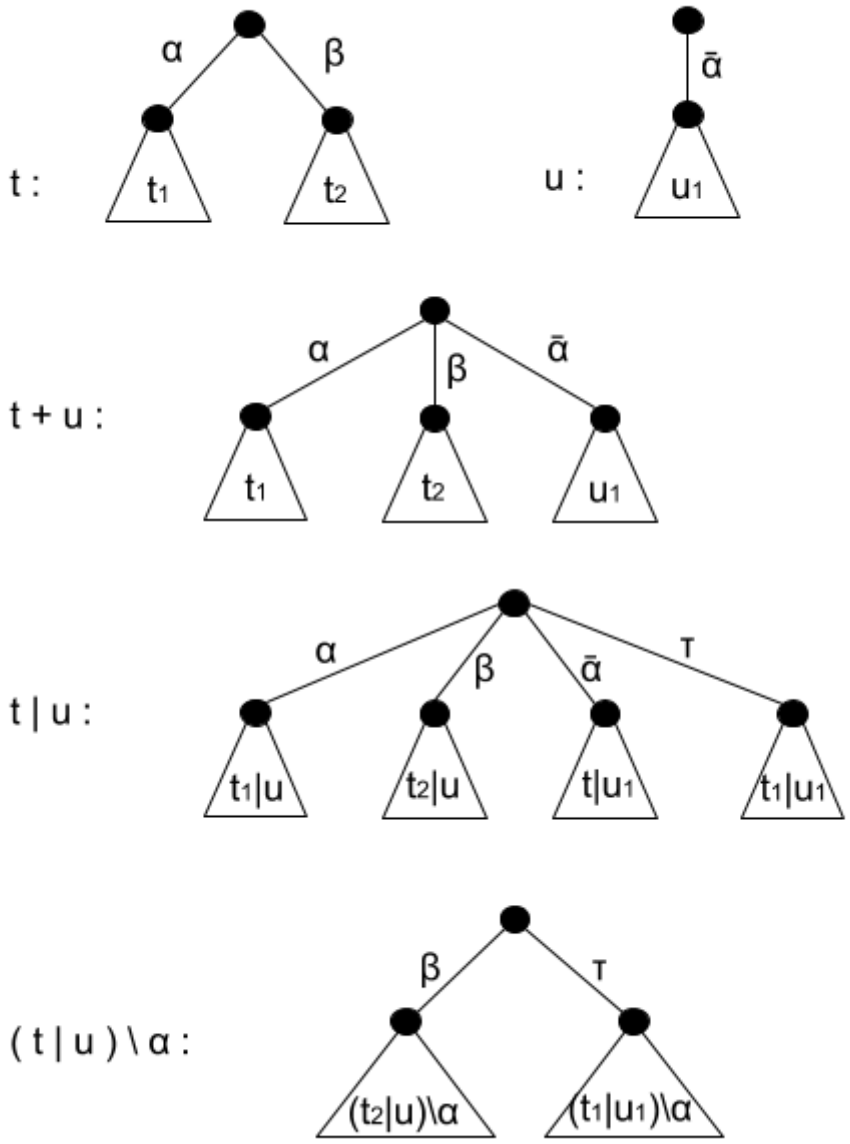
Pro grafické vyjádření procesů v algebře CCS jsou v [10] používány tzv. **synchronizační stromy**, které jsou obdobou přechodového diagramu. Synchronizační strom je neorientovaný strom konečného větvení s jediným kořenovým uzlem, jehož každá hrana je označena symbolem vstupní, výstupní nebo interní akce.

Prázdný proces NIL se značí jako strom tvořený jediným uzlem $\bullet$. Nechť synchronizační strom t reprezentuje libovolný proces. Aplikací operace komunikační akce λ na tento proces získáme strom t' (viz obrázek 2.11).



Obrázek 2.11: Grafické znázornění procesů pomocí synchronizačních stromů ($t' = \lambda.t$).

Na následujícím obrázku 2.12 jsou dále znázorněny operace sčítání, paralelní kompozice a restrikce nad synchronizačními stromy t a u.



Obrázek 2.12: Grafické znázornění operace sčítání, paralelní kompozice a restrikce nad synchronizačními stromy t a u.

Kapitola 3

π-kalkul

Procesní algebra π-kalkul, někdy též označována jako kalkul mobilních procesů, byla definována počátkem osmdesátých let dvacátého století jako rozšíření procesní algebry CCS popsané v kapitole 2.2.2. Jedná se o obecný výpočetní model určený pro formální popis interaktivních mobilních systémů a analýzu jejich vlastností a chování. Informace uvedené v této kapitole byly čerpány z [20] a [13].

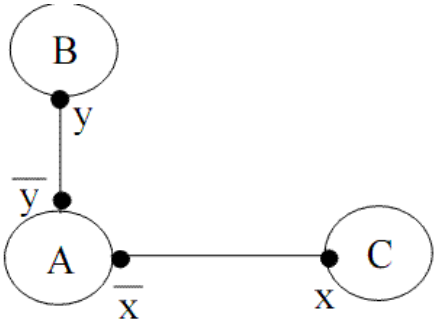
Mobilní systémy jsou postaveny na myšlence vzájemné interakce souběžných komponent prostřednictvím zasílání zpráv, přičemž tyto komponenty i vazby mezi nimi mohou za běhu systému dynamicky vznikat, zanikat nebo měnit své uspořádání. Příkladem takových systémů mohou být síťě mobilních komunikačních zařízení, počítačové sítě, WWW nebo i programy zapsané v objektově-orientovaném jazyce.

Z uvedené definice mobilních systémů vyplývá, že bereme-li v úvahu pouze virtuální, nikoliv fyzický, prstor, můžeme rozlišovat dva základní **typy mobility**. Prvním z nich je mobilita samotných komponent, druhým pak mobilita spojení (tj. vazeb mezi nimi). Zatímco π-kalkul prvního řádu se zabývá výhradně mobilitou vazeb, π-kalkul vyššího řádu je schopen modelovat i pohyb jednotlivých komponent ve virtuálním prostoru.

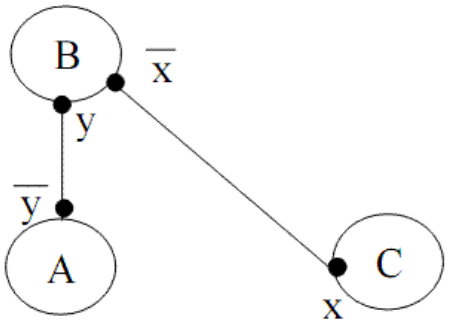
Stejně jako v případě procesní algebry CCS, tak i v případě π-kalkulu, jsou základními entitami **procesy** reprezentující komponenty systému a **jména** označující vazby mezi nimi.

Oproti CCS však π-kalkul zavádí nový způsob komunikace, a to komunikaci s **předáváním jmen**, jako součást akcí prováděných procesy. Základní π-kalkul nedefinuje datové typy, předávanou hodnotou tak může být cokoliv, zpravidla se však jedná o jméno, které můžeme chápat jako vazbu či referenci na komunikační kanál k určitému procesu. Podle toho, zda je předávána pouze jediná hodnota, anebo více, dělíme kalkul na monadický a polyadický. Přijetím jednoho nebo více jmen získává proces možnost interagovat i s procesy, které pro něj byly dosud nedostupné. Struktura systému daná vzájemným propojením komponent systému, se tak může interakcí komponent v čase měnit, což odpovídá definici mobilního systému.

Na obrázku 3.1 je znázorněna síť tří procesů A, B, C a vazeb mezi nimi. S procesem C je schopen iniciálně komunikovat pouze proces A prostřednictvím komunikačního kanálu x. Proces A však může odeslat název tohoto kanálu skrz komunikační kanál y a předat tak referenci na proces C procesu B. Sám se navíc této reference může vzdát. Výsledná konfigurace sítě po provedení uvedené akce je znázorněna na obrázku 3.2.



Obrázek 3.1: Výchozí struktura systému tvořeného procesy A, B, C a komunikačními kanály x a y. Přístup k C má pouze proces A.



Obrázek 3.2: Znázornění struktury systému z obrázku 3.1 po provedení komunikační akce, v důsledku které došlo k mobilitě kanálu x.

Dalším rozdílem procesní algebry π-kalkul oproti CCS je zavedení nového operátoru **replikace**, který společně s předávání jmen prostřednictvím zasílaných zpráv, umožňuje modelovat nekonečné chování a rekurzi procesů. Základní π-kalkul se tak teoreticky obejde bez definičních rovnic a identifikátorů procesů. Sémantika operátoru replikace bude detailněji popsána v kapitole 3.1.

π-kalkul je obzvlášť vhodný pro popis chování komplexních konkurentních systémů tvořených větším množstvím vzájemně komunikujících komponent, kdy každá z těchto komponent může být tvořena dalšími podkomponentami. Je nutné si uvědomit, že každá komponenta uvnitř systému představuje sama o sobě systém, jehož chování je možné specifikovat prostřednictvím výrazu π-kalkulu. To odpovídá indukční definici procesu, kdy každý proces popsaný výrazem procesní algebry je s využitím příslušných operátorů vytvořen jako kompozice jednodušších procesů (výrazů). Aplikací příslušných odvozovacých pravidel na jednotlivé výrazy, pak můžeme dané výrazy upravovat a zkoumat akce, které jsou popsané části systému, na libovolné úrovni zanoření, schopny provádět.

Chování systému, popsaného prostřednictvím výrazů procesní algebry π-kalkul, je možné analyzovat ze dvou různých úhlů pohledu. Prvním z nich je popis vývoje systému nezávisle na prostředí, pro který využíváme **relaci redukce**, popisující aktivitu uvnitř systému (tj. vzájemné interakce komponent systému). Relace redukce bude blíže popsána v kapitole 3.3. Druhý pohled se poté zaměřuje nejen na aktivitu uvnitř systému (tj. interní akce), ale i na možné interakce systému s prostředím (tj. vstupní a výstupní akce). V tomto případě je klíčovým prvkem **relace přechodu**, která určuje jak se systém může vyvíjet pro-

váděním konkrétních akcí. Relace přechodu jakožto rozšíření relace redukce bude popsána v kapitole 3.4

π-kalkul představuje **univerzální výpočetní model** o stejné výpočetní síle jako Turingův stroj. Je možné jej tak použít nejen pro popis chování klasických konkurentních systémů, ale také pro specifikaci datových struktur, objektů, funkcí, protokolů či jiných výpoečtních modelů (např. λ-kalkulu). Dva konkrétní příklady aplikace π-kalkulu budou uvedeny v kapitole 3.5.

3.1 Syntax a sémantika

V této kapitole bude popsána jak syntax jazyka π-kalkulu, tak i význam jednotlivých konstrukcí. Notace výrazů použitá v této i následujících kapitolách práce odpovídá notaci použité v [20].

Termy jazyka - **procesy** značíme zpravidla velkými písmeny latinské abecedy (P, P', P_i,...). **Jména**, která slouží nejen jako prostředek komunikace mezi procesy, ale také jako přenášená hodnota, značíme malými písmeny latinské abecedy (a, b, x,...).

Procesy se vyvíjejí prostřednictvím **akcí**. Schopnost procesu vykonat akci se značí pomocí operátoru prefixu π. Význam tohoto operátoru je takový, že proces popsaný výrazem $\pi.P$ musí nejprve vykonat akci π a teprve poté může pokračovat jako proces P. Celkem rozlišujeme 4 typy prefixů:

$$\pi := x(y) \mid \bar{x}\langle y\rangle \mid \tau \mid [x = y]\pi$$

- **vstupní prefix** - Proces $x(y).P$ může přijmout jakékoliv jméno z na vstupním portu komunikačního kanálu (jména) x a dále pokračovat jako proces P, v němž jsou všechny volné výskyty jména y nahrazeny za přijaté jméno z. Takovýto proces obvykle značíme $P\{z/y\}$.
- **výstupní prefix** - Proces $\bar{x}\langle y\rangle.P$ musí nejprve odeslat jméno y na výstupním portu komunikačního kanálu (jména) x a teprve poté může pokračovat jako proces P. Výstupní porty jsou značeny jménem kanálu s nadtržením.
- **nepozorovatelný prefix** - Proces $\tau.P$ může provést nepozorovatelnou interní akci a následně pokračovat jako proces P.
- **prefix shody** - Proces $[x = y]\pi.P$ může pokračovat jako proces $\pi.P$ pouze v případě, že x a y odpovídají stejnému jménu. V opačném případě nemůže dělat nic.

Mezi základní operace π-kalkulu dále řadíme sčítání (výběr), paralelní kompozici, restrikci a replikaci. Nechť P, P' označuje procesy, M, M' sumy procesů a z libovolné jméno, pak výrazy π-kalkulu můžeme definovat následovně:

$$P := M \mid P|P' \mid \nu z P \mid !P$$
$$M := 0 \mid \pi.P \mid M + M'$$

Jediným atomickým procesem π-kalkulu je proces **0**, který označuje **prázdnou sumu**. Jedná se o proces, který není schopen provádět žádné akce, ani se dále vyvíjet, tvoří však základ každého procesu. Je-li proces 0 uvozen prefixem π, jedná se o proces, který provede akci odpovídající příslušnému prefixu a následně skončí. U takovýchto procesů není nezbytně nutné symbol 0 ve výrazu explicitně uvádět, například proces $x(y).0$ tak můžeme

při zachování významu zjednodušeně zapsat jako $x(y)$. Pro prázdný proces dále platí axiomy 3.1.

$$\begin{aligned} M + 0 &= M \\ P \mid 0 &= P \\ \nu z 0 &= 0 \end{aligned} \tag{3.1}$$

Operace **sčítání** umožňuje modelovat výběr procesu. Proces $M+M'$ se tak může vyvíjet buď jako podproces M, nebo podproces M'. Podprces, který se ve výběru neuplatní, nemá na další vývoj procesu žádný vliv a je z výrazu vypuštěn.

Operace **paralelní kompozice** modeluje nezávislý běh dvou a více procesů, které mohou vzájemně interagovat prostřednictvím sdílených jmen. U procesu $x(y) \mid \bar{x}(z)$ tak existují 3 možnosti vývoje, kdy tento proces může:

- přijmout jakékoliv jméno na vstupním portu x,
- odeslat jméno z na výstupním portu $\bar{x}$,
- vyvinout se nepozorovaně jako efekt vzájemné interakce podkomponent prostřednictvím sdíleného jména x.

Operace **restrikce** omezuje rozsah specifikovaného jména pouze na určitý proces. Říkáme, že výskyt tohoto jména je v procesu vázaný. Komponenty daného procesu pak mohou toto jméno používat ke vzájemné interakci mezi sebou, avšak nikoliv jako prostředek komunikace s dalšími procesy. Například proces $\nu x(x(y) \mid \bar{x}(z))$ se tak může vyvíjet pouze jediným způsobem, a to interakcí vnitřních komponent prostřednictvím sdíleného jména x. Obecně platí, že rozsah omezení se může interakcí změnit. Mějme například dva procesy definované následovně:

$$\begin{aligned} P &\stackrel{\text{def}}{=} \nu z(\bar{x}\langle z\rangle.P') \\ Q &\stackrel{\text{def}}{=} x(y).Q' \end{aligned}$$

Interakcí těchto procesů prostřednictvím x tak dojde k rozšíření platnosti omezení jména z i na proces Q'. V dalším textu této práce budeme daný stav označovat také jako *únik jména z z rozsahu* (anglicky *scope extrusion*).

$$P \mid Q \xrightarrow{\tau} \nu z(P' \mid Q')$$

Operace **replikace** představuje nekonečnou paralelní kompozici procesů. Platí, že

$$!P = P \mid !P = P \mid P \mid P \mid ... \mid !P$$

Replikace tak umožňuje vyjádřit nekonečné chování procesů. Definujme například následující procesy P a Q:

$$\begin{aligned} P &\stackrel{\text{def}}{=} !x(z).\bar{y}\langle z\rangle \\ Q &\stackrel{\text{def}}{=} !x(z).\,!\bar{y}\langle z\rangle \end{aligned}$$

Proces P je schopen do nekonečna přijímat jména na vstupním portu x a každé takto přijaté jméno následně odeslat na výstupním portu komunikačního kanálu y. Naproti tomu, Q představuje obdobný proces schopný do nekonečna přijímat jména na vstupním portu x,

avšak na rozdíl od P, každé takto přijaté jméno může být opakovaně odesíláno na výstupním portu y.

Operátor přeznačení, který je jedním ze základních operátorů procesní algebry CCS, je v případě π-kalkulu nahrazen **substituční funkcí** σ. Substituční funkce je definována jako zobrazení na množině všech jmen. Nechť $\{x_1, x_2, ..., x_n\}$ představuje množinu jmen, které mají být nahrazeny odpovídajícími jmény z množiny $\{y_1, y_2, ..., y_n\}$, pak $\sigma(x_i) = y_i$ obvykle zapisujeme ve tvaru $\{y_1, y_2, ..., y_n/x_1, x_2, ..., x_n\}$. Pro všechna ostatní jména $z \notin \{x_1, x_2, ..., x_n\}$ platí, že $\sigma(z) = z$.

Dříve než rozšíříme aplikaci substituční funkce na procesy, definujme zde pojem volného a vázaného výskytu jména v procesu.

Definice 3.1.1. *Výskyt jména z v procesu P je **vázaný**, pokud byl na daný proces aplikován operátor restrikce νzP nebo operátor vstupního prefixu ve tvaru $x(z).P$.*

Definice 3.1.2. *Výskyt jména je **volný** právě tehdy, když není vázaný.*

Volná jména jsou obecně taková, která se mohou účastnit vnějších akcí, tedy chce-li libovolný proces P odeslat jméno x, nebo chce-li toto jméno použít jako komunikační kanál, ať už pro vstup či výstup, musí platit, že $x \in fn(P)$, kde $fn(P)$ značí množinu všech volných jmen procesu P.

Aplikace substituční funkce na proces $P\{w/z\}$ umožňuje nahradit všechny volné výskyty jména z v P za odpovídající jméno $w = \sigma(z)$, které se v daném výrazu P nevyskytuje. Každý podvýraz procesu P ve tvaru $x(z).Q$ je pak nutné nahradit výrazem $x(w).Q\{w/z\}$ a obdobně i každý podvýraz νzQ výrazem $\nu wQ\{w/z\}$. Procesy P a Q označujeme jako **α-konvertibilní**, pokud se liší pouze v názvech vázaných jmen.

Vztah substituční funkce k ostatním operátorům je vyjádřen skupinou rovnic 3.2:

$$\begin{aligned} 0\sigma &\stackrel{\text{def}}{=} 0 \\ (\pi.P)\sigma &\stackrel{\text{def}}{=} \pi\sigma.P\sigma \\ (P + P')\sigma &\stackrel{\text{def}}{=} P\sigma + P'\sigma \\ (P \mid P')\sigma &\stackrel{\text{def}}{=} P\sigma \mid P'\sigma \\ (\nu zP)\sigma &\stackrel{\text{def}}{=} \nu zP\sigma \\ (!P)\sigma &\stackrel{\text{def}}{=} !P\sigma \end{aligned} \tag{3.2}$$

Priorita operátorů a funkce substituce je v π-kalkulu definována následujícími pravidly:

1. operátor prefixu, restrikce a replikace se váže pevněji než operátor kompozice
2. operátor prefixu se váže pevněji než operátorem sčítání
3. substituce má přednost před všemi operátory

Priorita operátorů určuje postup při vyhodnocování výrazů. Pro nestandardní pořadí vyhodnocování výrazů nebo zjednodušení interpretace se používají kulaté závorky.

Poznámka. Vzájemný vztah operátoru sčítání a paralelní kompozice nebyl v žádné z prostudovaných publikací vyjádřen explicitním pravidlem. R. Milner však ve své knize [13] píše: "*Vyskytuje-li se ve výrazu suma více než jednoho termu uvnitř restrikce nebo kompozice, zapisujeme ji do závorek následovně:* $(a.P + b.Q) \mid c.R$". Z uvedeného tvrzení je možné usuzovat, že operátor restrikce i kompozice se oba váží pevněji než operátor sčítání a výraz $(a.0 \mid b.0 + c.0 \mid d.0)$ by tedy měl být interpretován jako $((a.0 \mid b.0) + (c.0 \mid d.0))$.

3.2 Abstrakce a aplikace

Dalším způsobem vázání jmen v π-kalkulu je kromě operátoru vstupního prefixu a operátoru restrikce deinice parametrizovaného procesu. Pro definci takového prcesu se zpravidla používá zápis $A(\vec{a}) \stackrel{\text{def}}{=} Q_A$, kde A je identifikátor procesu a $\vec{a}$ seznam všech volných jmen ve výrazu Q_A.

Pro sjednocení všech možných způsobů vázání jmen x_i v procesu P byl zaveden koncept **abstrakce**. Abstrakci arity $n \geq 0$ zapisujeme jako $(x_1).\cdots(x_n).P$ nebo zjednodušeně jako $(x_1 \cdots x_n).P$. Samotný proces P bez parametrů je pak abstrakcí s aritou 0. Vstupní prefix procesu $y(x).P$ můžeme chápat jako abstrakci procesu P na jméně y. Tabulka 3.1 uvádí všechny způsoby vázání jmen v π-kalkulu (v levém sloupci) společně s odpovídajícím zápisem abstrakce (v pravém sloupci).

$x(\vec{a}).P$	$x((\vec{a}).P)$
νaP	$\nu((a).P)$
$A(\vec{a}) \stackrel{\text{def}}{=} P$	$A \stackrel{\text{def}}{=} (\vec{a}).P$

Tabulka 3.1: Tabulka mapující zápis navázání jména na odpovídající zápis abstrakce.

Mějme abstrakci $F \stackrel{\text{def}}{=} (\vec{x}).P$ s aritou n a n-tici jmen $\vec{y}$. Aplikaci abstrakce F na $\vec{y}$ značíme $F\langle\vec{y}\rangle$. Výsledkem této aplikace je proces $P\{\vec{y}/\vec{x}\}$, který je instancí abstrakce F. Abstrakci si tak můžeme představit jako třídu procesů a aplikaci jako instanciaci této třídy. Výše uvedený způsob aplikace se někdy označuje jako takzvaná pseudo-aplikace. Druhým typem je konstantní aplikace, která umožňuje popisovat rekurzivně definované procesy. Nechť K je rekurzivní definice konstanty procesu značená výrazem $K \triangleq (\vec{x}).P$, kde $\vec{x}$ obsahuje všechna jména která mají volný výskyt v P. Konstatní aplikace se pak značí $K\lfloor\vec{a}\rfloor$.

3.3 Relace redukce

Výpočetní krok procesu ve smyslu vzájemné interakce jeho podprocesů, můžeme formálně definovat pomocí relace redukce $\rightarrow$. Proces P je možné redukovat na proces Q (značíme $P \rightarrow Q$) v případě, že P je tvořeno paralelní kompozicí alespoň dvou podprocesů, které mohou provést interní komunikační akci prostřednictvím sdíleného kanálu a následně pokračovat ve formě odpovídajících podprocesů jejichž paralelní kompozice tvoří proces Q.

Pro definici redukce je klíčová relace **strukturální kongruence** značená symbolem $\equiv$, která umožňuje upravovat strukturu výrazů při zachování jejich významu. Strukturální kongruenci je možné chápat jako relaci mezi procesy, které jsou zapsány rozdílným způsobem, avšak významově jsou totožné. Jsou-li dva procesy v této relaci, lze kterýkoliv z nich transformovat na druhý použitím axiomů strukturální kongruence 3.3.

$$
\begin{aligned}
[x = x]\pi.P &\equiv \pi.P \\
M_1 + (M_2 + M_3) &\equiv (M_1 + M_2) + M_3 \\
M_1 + M_2 &\equiv M_2 + M_1 \\
M + 0 &\equiv M \\
P_1 \mid (P_2 \mid P_3) &\equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 &\equiv P_2 \mid P_1 \\
P \mid 0 &\equiv P \\
\nu z \nu w P &\equiv \nu w \nu z P \\
\nu z 0 &\equiv 0 \\
\nu z (P_1 \mid P_2) &\equiv P_1 \mid \nu z P_2, \quad \text{if} \quad z \notin fn(P_1) \\
!P &\equiv P \mid !P
\end{aligned}
\tag{3.3}
$$

Relace redukce je definována množinou odvozovacích pravidel uvedených níže. Procesy P a Q jsou v relaci redukce právě tehdy, je-li $P \to Q$ možno odvodit z uvedených pravidel.

$$R - INTER: \quad \overline{(\bar{x}\langle y\rangle.P_1 + M_1) \mid (x(z).P_2 + M_2) \to P_1 \mid P_2\{y/z\}}$$

$$R - TAU: \quad \overline{\tau.P + M \to P}$$

$$R - PAR: \quad \frac{P \to P'}{P \mid Q \to P' \mid Q}$$

$$R - RES: \quad \frac{P \to P'}{\nu z P \to \nu z P'}$$

$$R - STRUCT: \quad \frac{P_1 \equiv P_2 \to P_2' \equiv P_1'}{P_1 \to P_1'}$$

Základním axiomem redukce je pravidlo R-INTER. Proces na levé straně pravidla je tvořen paralelní kompozicí dvou komponent, kdy první z nich umožňuje odeslat jméno y prostřednictvím jména x a druhá přijmout libovolné jméno použitím stejného jména x. Podprocesy uvozené odpovídajícími si dvojicemi vsupních a výstupních akcí, které využívají stejný komunikační kanál, tvoří takzvaný **redex**. Existuje-li ve výrazu více redexů, existuje i více možností, jak tento výraz redukovat.

Druhým axiomem je pravidlo R-TAU. Prefix τ reprezentuje interakci uvnitř procesu P, jejíž původ však není explicitně vyjádřen.

Pavidlo R-PAR specifikuje, že pokud nastane vzájemná komunikace mezi dvěma podprocesy procesu P, pak ke stejné komunikaci dojde i v případě, že je proces P umístěn do paralelní kompozice s libovolným procesem Q. Obdobně pravidlo R-RES určuje, že ke stejné komunikaci dojde i pokud je na daný proces aplikován operátor restrikce νz.

Pravidlo R-STRUCT poté umožňuje používat výše uvedená pravidla i pro procesy, které se mohou lišit zápisem, avšak jsou strukturálně kongruentní s procesem uvedeným v pravidle.

Redukční derivaci můžeme vizualizovat formou stromu, jehož listy jsou instancemi pravidel R-INTER a R-TAU a jehož kořenem je výraz, který se pokoušíme aplikací pravidel odvodit. Na obrázku níže je graficky znázorněn příklad postupné aplikace redukčních pravidel za účelem odvození redukce procesu $P = \nu a((a.Q_1 + b.Q_2) \mid \bar{a}) \mid (\bar{b}.R_1 + \bar{a}.R_2)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(a.Q_1 + b.Q_2) \mid \bar{a} \to Q_1 \mid 0}\,R{-}INTER}{(a.Q_1 + b.Q_2) \mid \bar{a} \to Q_1}\,R{-}STRUCT}{\nu a((a.Q_1 + b.Q_2) \mid \bar{a}) \to \nu a Q_1}\,R{-}RES}{\nu a((a.Q_1 + b.Q_2) \mid \bar{a}) \mid (\bar{b}.R_1 + \bar{a}.R_2) \to \nu a Q_1 \mid (\bar{b}.R_1 + \bar{a}.R_2)}\,R{-}PAR$$

3.4 Relace přechodu

Pro porozumnění chování systému jako celku je obvykle výhodné analyzovat chování jednotlivých částí tohoto systému zvlášť, tedy zkoumat jakých akcí jsou tyto části schopné. K tomuto účelu slouží relace přechodu $\xrightarrow{\alpha}$, která je rozšířením relace redukce definované v předchozí kapitole. Na rozdíl od relace redukce, která popisuje aktivitu pouze uvnitř procesu, relace přechodu vyjadřuje schopnost procesu provádět vstupně-výstupní akce a reagovat tak i s dalšími konkurentně běžícími procesy. Zápis $P \xrightarrow{\alpha} P'$ znamená, že proces P je schopen přejít do stavu P' provedením akce α. Dva konkurentně běžící procesy mohou vzájemně interagovat v případě, že jeden z nich je schopen provést libovolnou vstupní akci a druhý z nich komplementární výstupní akci. Celkem rozlišujeme následující typy akcí:

$$\alpha := \bar{x}y \mid xy \mid \bar{x}(z) \mid \tau$$

První z nich reprezentuje odeslání jména y prostřednictvím jména x, druhá přijetí libovolného jména y skrz kanál x, třetí odeslání vázaného jména z prostřednictvím x a poslední interní akci. V případě, že $\alpha = \tau$, pak relace přechodu odpovídá relaci redukce.

Relace přechodu je rovněž definována skupinou odvozovacích pravidel:

$$OUT: \quad \frac{}{\bar{x}\langle y\rangle.P \xrightarrow{\bar{x}y} P}$$

$$INP: \quad \frac{}{x(y).P \xrightarrow{xy} P\{y/z\}}$$

$$TAU: \quad \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$MAT: \quad \frac{\tau.P \xrightarrow{\alpha} P'}{[x = x]\tau.P \xrightarrow{\alpha} P'}$$

$$SUM - L: \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$PAR - L: \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad bn(\alpha) \cap fn(Q) = \emptyset$$

$$COMM - L: \quad \frac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$CLOSE - L: \quad \frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} \nu z(P' \mid Q')} \qquad z \notin fn(Q)$$

$$RES: \quad \frac{P \xrightarrow{\alpha} P'}{\nu z P \xrightarrow{\alpha} \nu z P'} \qquad z \notin n(\alpha)$$

$$OPEN: \quad \frac{P \xrightarrow{\bar{x}z} P'}{\nu z P \xrightarrow{\bar{x}(z)} P'} \qquad z \neq x$$

$$REP - ACT: \quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$REP - COMM: \quad \frac{P \xrightarrow{\bar{x}y} P' \quad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$REP - CLOSE: \quad \frac{P \xrightarrow{\bar{x}(z)} P' \quad P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} (\nu z(P' \mid P'')) \mid !P} \qquad z \notin fn(P)$$

Ve přehledu pravidel výše nejsou uvedena pravidla SUM-R, PAR-R, COMM-R a CLOSE-R, která jsou pouze symetrickým tvarem pravidel SUM-L, PAR-L, COMM-L a CLOSE-L.

3.5 Příklady

V této kapitole bude na dvou příkladech ukázáno použití procesní algebry π-kalkul pro specifikaci chování systémů. Prvním bude specifikace datové struktury elastického bufferu, druhým pak jednoduchý příklad byznys procesu - objednávka přes internetový obchod.

3.5.1 Buffer

Buffer o kapacitě 1 můžeme chápat jako buňku B, která má jeden vstup a jeden výstup.

$$B(i,o) \stackrel{\text{def}}{=} i(x).\bar{o}\langle x\rangle.B\langle i,o\rangle$$

Buffer o kapacitě n je pak zřetězením n takovýchto buněk, kdy vstup každé z buněk (kromě první) je navázán na výstup levé sousední buňky.

$$Buff_n \stackrel{\text{def}}{=} (\nu\ o_1,...,o_n)(B\langle i,o_1\rangle \mid B\langle o_1,o_2\rangle \mid ... \mid B\langle o_{n-1},o\rangle)$$

Neomezený buffer UB můžeme navrhnout tak, že při kadém vstupu vytvoříme novou buňku C, která bude udržovat přijatou hodnotou:

$$\begin{aligned} UB(i,o) &\stackrel{\text{def}}{=} i(x).(\nu m)(UB\langle i,m\rangle \mid C\langle x,m,o\rangle) \\ B(i,o) &\stackrel{\text{def}}{=} i(x).C\langle x,i,o\rangle \\ C(x,i,o) &\stackrel{\text{def}}{=} \bar{o}\langle x\rangle.B\langle i,o\rangle \end{aligned}$$

Problém tohto bufferu však spočívá v tom, že při výstupu nejsou tyto buňky odstraněny a tento buffer tak může pouze růst. Vylepšením je elastický buffer EB, který může nejen zvětšovat, ale i zmenšovat svou kapacitu. Princip spočívá v tom, že je-li poslední buňka v řadě obsahující hodnotu (RC) vyprázdněna, předá odkaz na svůj výstupní port své levé sousední buňce a z řetězce se odebere (transformuje se na prázdný proces 0). Elastický buffer tedy můžeme definovat následovně:

$$\begin{aligned} EB(i,o) &\stackrel{\text{def}}{=} i(x).(\nu m)(B\langle i,m\rangle \mid RC\langle x,m,o\rangle) \\ B(i,m) &\stackrel{\text{def}}{=} i(x).(\nu n)(B\langle i,n\rangle \mid C\langle x,n,m\rangle) + m(o).EB\langle i,o\rangle \\ C(x,n,m) &\stackrel{\text{def}}{=} m(o).RC\langle x,n,o\rangle \\ RC(x,n,o) &\stackrel{\text{def}}{=} \bar{o}\langle x\rangle.\overline{n}\langle o\rangle.0 \end{aligned}$$

3.5.2 E-shop

Předpokládejme pro jednoduchost, že systém, který chceme modelovat, je tvořený následujícími komponentami: zákazník $Client$, internetový obchod $EShop$, prodejce $Vendor$ a banka $Bank$.

Zákazník provede objednávku zboží přes internetový obchod. Kromě samotné objednávky ord zadá své identifikační údaje id a adresu $addr$. Následně obdrží informace potřebné pro provedení platby a odkaz pro zrušení objednávky. Klient se může rozhodnout, zda za zboží zaplatí a bude očekávat jeho přijetí na zadané adrese, nebo svou objednávku zruší.

$$\begin{aligned} Client(shop, id, addr) \stackrel{\text{def}}{=} &(\nu\, ord)(\overline{shop}\langle id, ord, addr\rangle.id(bank, bill, confirm, cancel) \\ &.((\nu\, acc)(\overline{bank}\langle acc, bill, confirm\rangle.addr(goods)) + \overline{cancel}) \end{aligned}$$

Internetový obchod vystupuje jako prostředník mezi zákazníkem a prodejcem. Přijme objednávku od klienta, kterému následně zašle informace o platbě. V případě, že obdrží potvrzení z banky o provedení platby, předá obchodníkovi objednávku ke zpracování.

$$\begin{aligned} EShop(web, vend, bank) \stackrel{\text{def}}{=} &web(client, order, addr) \\ &.(\nu\, invoice, conf, cancel)(\overline{client}\langle bank, invoice, conf, canc\rangle \\ &.(conf.\overline{prod}\langle order, addr\rangle.EShop\langle web, vend, bank\rangle \\ &+ canc.EShop\langle web, vend, bank\rangle)) \end{aligned}$$

Obdrží-li prodejce informace o objednávce, odešle objednané zboží zákazníkovi.

$$Vendor(eshop) \stackrel{\text{def}}{=} eshop(order, addr).(\nu\, goods)(\overline{addr}\langle goods\rangle.Vendor\langle eshop\rangle)$$

Banka zpracuje platbu a zašle potvrzení internetovému obchodu.

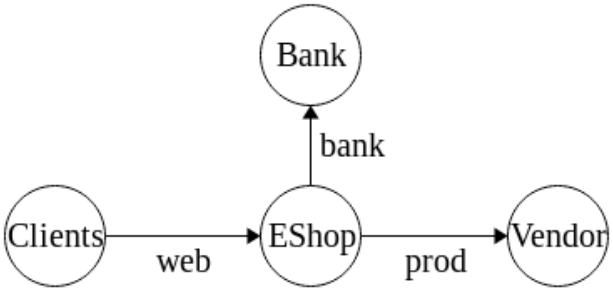
$$Bank(bank) \stackrel{\text{def}}{=} bank(account, bill, confirm).\overline{confirm}.Bank\langle bank\rangle$$

Za účelem modelování nekonečného běhu systému, je potřeba specifikovat ještě pomocný proces $Clients$, který bude fungovat jako generátor klientů:

$$Clients(eshop) \stackrel{\text{def}}{=} (\nu\, id, addr)(Client\langle eshop, id, addr\rangle \mid Clients\langle eshop\rangle)$$

Výsledný systém zobrazený na obrázku 3.3 je paralelní kompozicí výše definovaných procesů.

$$\begin{aligned} System(web, prod, bank) \stackrel{\text{def}}{=} &Clients\langle web\rangle \mid EShop\langle web, prod, bank\rangle \mid \\ &Vendor\langle prod\rangle \mid Bank\langle bank\rangle \end{aligned}$$



Obrázek 3.3: Grafické znázornění struktury modelovaného systému včetně komunikačních kanálů.

Kapitola 4

Praktické aplikace π-kalkulu

π-kalkul nalezl praktické uplatnění jako základ několika programovacích jazyků určených pro konkurentní programování. Mezi hlavní představitele patří jazyk **Pict** (Pierce, Turner, 1997) a jazyk **Occam-π**, jenž je rozšířením jazyka occam. Za příbuzné programovací jazyky poté můžeme označit i **Join** (Fournet, Maranget, 1997) nebo jazyk **TyCO** (Vasconcelos, Bastos, 1998). π-kalkul zformoval rovněž teoretické základy jazyka **BPML** určeného pro modelování byznys procesů. V článku [9] pak popsal Liwu Li možnost interpretace π-kalkulu v jazyce Java s použitím mezivláknové komunikace.

Další uplatnění π-kalkulu bychom mohli nalézt v oblasti počítačové bezpečnosti, kde jako příklad můžeme uvést **Spi-kalkul** [1], jenž rozšířil notaci π-kalkulu o primitiva šifrování a dešifrování a stal se tak vhodným formalismem pro popis a analýzu kryptografických protokolů. Na jeho základech bylo vytvořeno několik verifikačních nástrojů, z nichž můžeme uvést jako příklad **ProVerif**[1], jehož autorem je Bruno Blanchet. ProVerif a umožňuje uživateli prostřednictvím příkazové řádky provést analýzu bezpečnosti kryptografického protokolu zapsaného v typovaném jazyce π-kalkul.

Kromě nástroje ProVerif však existuje ještě řada obecných verifikačních nástrojů, které umožňují simulovat a analyzovat chování systémů popsaných výrazy π-kalkulu [19].

Prvním z nich je **Mobility Workbench**[2] zkráceně MWB, který byl vytvořen ve funkcionálním jazyce SML čtveřicí autorů Victor, Moller, Eriksson a Dam. Jedná se o nástroj umožňující interaktivní simulaci systémů popsaných výrazy π-kalkulu, včetně ověřování modelu a analýzy některých vlastností jako například možnosti uváznutí. Mimo to umožňuje ověřit existenci různých typů relací ekvivalence mezi zadanými procesy.

Obdobným nástrojem je **Another/Advanced Bisimulation Checker**[3] zkráceně ABC, implementovaný ve funkcionálním jazyce OCaml Sébastienem Briaisem. Na rozdíl od MWB nepodporuje ověřování modelu ani hledání uváznutí, ale nabízí rozšířené možnosti pro zkoumání relace ekvivalence mezi procesy.

Posledním verifikačním nástrojem, jehož autorem je Matteo Mio, je **Pi-Calculus Equivalences Tester**[4] zkráceně PiET implementovaný ve funkcionálním programovacím jazyce Fresh O'Caml. Tento nástroj umožňuje porovnat procesy na 10 různých typů relací ekvivalence a jako jediný z výše uvedených nástrojů nabízí i jednoduché grafické uživatelské rozhraní implementované v jazyce Java (nikoliv však grafickou reprezentaci výrazů).

[1] http://prosecco.gforge.inria.fr/personal/bblanche/proverif
[2] http://www.it.uu.se/research/group/mobility/mwb
[3] http://sbriais.free.fr/tools/abc
[4] http://piet.sourceforge.net

4.1 Textové reprezentace výrazů

V této podkapitole budou popsány dva příklady textové reprezentace výrazů π-kalkulu. Prvním z nich bude textová reprezentace používaná v programovacím jazyce Pict verze 4.1 specifikovaná v [17], druhou pak textová reprezentace používaná pro specifikaci systémů v nástroji MWB podle [21].

Je nutné brát ohled na to, že Pict, jakožto plnohodnotný typovaný programovací jazyk, obsahuje některé konstrukce, které nejsou součástí původní specifikace procesní algebry π-kalkul. Naproti tomu syntax jazyka používaného v MWB je věrnějším obrazem syntaxe procesní algebry π-kalkul popsané v kapitole 3.1.

4.1.1 Pict

Program v jazyce Pict zapisujeme ve tvaru procesního výrazu uvozeného klíčovým slovem `run`.

`run` *Proc*

Procesní výraz může nabývat následujících podob:

Proc	:=	*Id* `!` *Val*	výstupní atom
		Id `?` *Abs*	vstupní prefix
		`(` `)`	prázdný proces
		`(` *Proc* `|` *Proc* `)`	paralelní kompozice procesů
		`(` *Dec* *Proc* `)`	lokální deklarace
		`if` *Val* `then` *Proc* `else` *Proc*	podmíněný výraz

Id značí identifikátor, kterým je v případě vstupního prefixu a výstupního atomu komunikační kanál. Hodnta *Val* označuje entitu, která může být odeslána skrze tento kanál. Může se obecně jednat o konstantu, proměnnou nebo záznam.

Val	:=	*Const*	konstanta
		Id	proměnná
		`[` *Label Val* ... *Label Val* `]`	záznam

Výstupní atom ve tvaru `x!y` značí odeslání hodnoty y prostřednictvím kanálu x. Po kanálu je však možné přenést i prázdný záznam `[]`. Výraz `x![]` pak reprezentuje výstupní synchronizační akci na kanálu x. Je-li přenášena konstanta, může se jednat o pravdivostní hodnotu `true` nebo `false`, číslo, znak nebo řetězcový literál.

Const	:=	`true`	pravda
		`false`	nepravda
		Int	číslo
		Char	znak
		String	řetězec

Vstupní proces je definován ve smyslu abstrakce procesu:

Abs	:=	*Pat* `=` *Proc*	abstrakce procesu

Prefix *Pat* se přitom může vyskytovat v některém z následujících tvarů:

Pat	:=	*Id* *RType*	prefix proměnné
		`[` *Label Pat ... Label Pat* `]`	prefix záznamu
		`_` *RType*	prefix vypuštěné proměnné
		Id *RType* `@` *Pat*	vícevrstvý prefix

Proces `x?z = P` tak nejprve přijme jakoukoliv hodnotu na komunikačním kanálu x a naváže ji na proměnnou z, následně pak pokračuje vykonáním podprocesu P.

Pict je staticky typovaný jazyk, což znamená, že přenášené hodnoty i komunikační kanály mohou mít přiřazen konkrétní typ.

RType	:=	⟨empty⟩	vypuštěná anotace typu
		Type	explicitní anotace typu

Kromě základních typů tento jazyk definuje i speciální typ pro komunikační kanál. Každý kanál je pak schopen přenášet pouze hodnoty daného typu.

Type	:=	`^`*Type*	vstupní/výstupní kanál
		Bool	typ pravdivostní hodnoty
		Int	typ čísla
		Char	typ znaku
		String	typ řetězce
		`[` *Label Type ... Label Type* `]`	záznam

Jednotlivé prvky v záznamu hodnot, typů i prefixů je možné pojmenovat. Pojmenování je volitelné a nemusí být vždy uvedeno.

Label	:=	⟨empty⟩	anonymní pole
		Id `=`	označené pole

Deklarací může být vytvoření nového komunikačního kanálu pomocí klíčového slova `new`, což odpovídá aplikaci operátoru restrikce, dále pak definice procesu nebo definici typu:

Dec	:=	`new` *Id* `:` *Type*	vytvoření kanálu
		`def` $Id_1\ Abs_1$ `and` ... $Id_n\ Abs_n$	rekurzivní definice pro $n \geq 1$
		`type` *Id* `=` *Type*	definice typu

4.1.2 MWB

Agenta (proces) v nástroji MWB definujeme následovně:

`agent` *Id Params* = *Proc*

Id je jednoznačný identifikátor procesu a *Params* seznam všech vstupních parametrů, neboli volných jmen vyskytujících se v procesním výrazu *Proc*. Seznam všech vstupních parametrů nemusí být uveden vůbec, pokud se ve výrazu nevyskytují žádná volná jména nebo se použije abstrakce (viz. dále). V opačném případě je ohraničen kulatými závorkami.

Params	:=	⟨empty⟩	bez parametrů
		`(` *nlist* `)`	seznam parametrů - volných jmen

Obdobně definujeme seznam vstupních parametrů, který je však ohraničen úhlovými závorkami.

Oparams	:=	⟨empty⟩	bez argumentů
		`<` *nlist* `>`	seznam argumentů

Jednotlivá jména v seznamech jsou oddělována pomocí čárky.

nlist	:=	*name*	jméno
		name `,` *nlist*	seznam jmen

Jméno musí vždy začínat malým písmenem latinské abecedy, přičemž poté může obsahovat speciální znaky _, $, ', číslice a písmena.

name	:=	`[a-z][a-zA-Z_$'0-9]*`	jméno

Identifikátor musí na rozdíl od jména začínat velkým počátečním písmenem.

Id	:=	`[A-Z][a-zA-Z_$'0-9]*`	identifikátor procesu

Syntaxe procesního výrazu je definována následující gramatikou:

Proc	:=	`0`	prázdný proces
		π`.`*Proc*	operátor prefixu
		`[`*name* `=` *name*`]`*Proc*	prefix shody
		Proc `|` *Proc*	paralelní kompozice procesů
		Proc `+` *Proc*	suma procesů
		Id Oparams	konkretizace procesu
		Id Params	alternativní zápis konkretizace procesu
		`(` `^` *nlist* `)` *Proc*	restrikce jmen
		`(` `\` *nlist* `)` *Proc*	vstupní abstrakce
		`[` *nlist* `]` *Proc*	výstupní abstrakce
		`(` *Proc* `)`	závorky pro vynucení priority vyhodnocení

Definice prefixu v MWB odpovídá syntaxi prefixu π-kalkulu s tím rozdílem, že nadtržení ve výstupním prefixu značíme pomocí apostrofu, $\bar{x}(y)$ tedy zapisujeme jako `'x<y>` a interní akci τ značíme `t`.

π	:=	*name Params*	vstupní akce
		`'`*name Oparams*	výstupní akce
		`t`	interní akce

MWB umožňuje používat abstrakci nejen pří definici procesů, ale i v případě vstupních a výstupních akcí. Následující dva zápisy jsou významově ekvivalentní:

```
P(x,y) = (^z)'x<y,z>.y(x,y).P<y,x>
P = (\x,y)(^z)'x.[y,z]y.(\x,y)P<y,x>
```

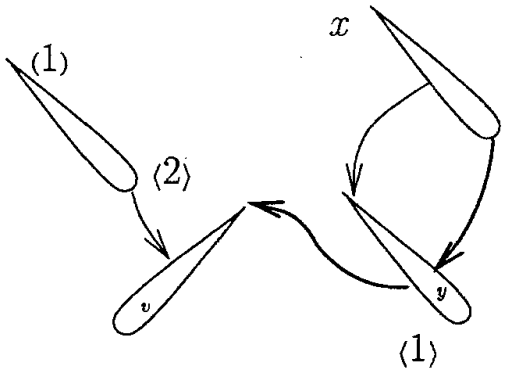
4.2 Grafické reprezentace výrazů

Všechny výše uvedené programovací jazyky i aplikace pracují výhradně s textovou reprezentací výrazů. Vizuální podobou π-kalkulu se však zabývalo několik odborníků, kteří své nápady zveřejnili formou odborných článků či publikací. V této podkapitole budou některé z těchto grafických reprezentací výrazů prezentovány.

4.2.1 π-nets

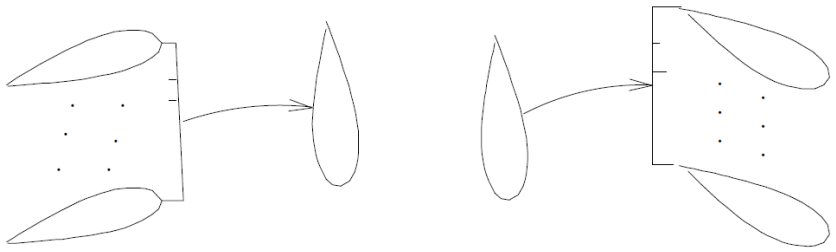
π-nets popsaný Robinem Milnerem v článku [12] je jedním z prvních pokusů o grafickou reprezentaci kalkulu akcí PIC. I přesto, že PIC postrádá některé konstrukce π-kalkulu (např. replikaci či operátor prefixu), v určitých aspektech s ním blízce souvisí. Základní entitou kalkulu akcí je akce $a : m \to n$, kde m je zdrojová arita a n cílová arita. Akce v PIC má tvar $a = (\vec{x})S\langle\vec{y}\rangle$, kde $\vec{x}$ je vektor vzájemně odlišných importovaných jmen ($|\vec{x}| = m$), $\vec{y}$ je vektor exportovaných jmen ($|\vec{y}| = n$) a S sekvence částic π-kalkulu, kterými může být částice vstupu $x(y)$, výstupu $\bar{x}\langle y\rangle$ nebo restrikce νx.

Na obrázku 4.1 je zobrazena grafická reprezentace akce $a = (z)[\, x(y)\ \bar{y}\langle x\rangle\ y(v)\ \bar{v}\langle z\rangle\ ]\langle yz\rangle$.



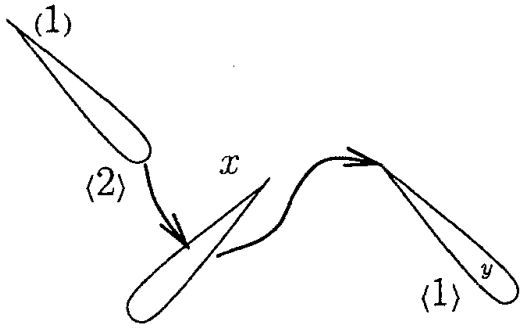
Obrázek 4.1: π-net akce $a = (z)[\, x(y)\ \bar{y}\langle x\rangle\ y(v)\ \bar{v}\langle z\rangle\ ]\langle yz\rangle$ — převzato z [12].

Jména v π-nets jsou reprezentovány asymetrickými uzly, u nichž rozlišujeme širší hlavu, tělo a zúžený konec. Uzly reprezentující volná jména jsou na užším konci označeny přímo (v tomto případě pouze x). Importovaná (exportovaná) jména jsou označena číselně na užším (respektive širším) konci. Orientované hrany reprezentují jednotlivé částice akce. V případě monadického kalkulu výstupní částice směřuje od širšího konce uzlu reprezentujícího přenášenou hodnotu ke středu uzlu reprezentujícího komunikační kanál. Vstupní částice pak směřuje od středu uzlu komunikačního kanálu k užšímu konci uzlu, který reprezentuje vstupní proměnnou. Příklad výstupní a vstupní částice u polyadického kalkulu je znázorněn na obrázku 4.2.

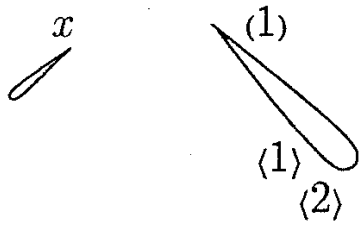


Obrázek 4.2: π-nets pro polyadický výstup a vstup — převzato z [11].

Dvojice hran, které se setkávají ve středu některého uzlu představují redex. Ten může být redukován odstraněním dané dvojice hran a sjednocením zdrojového a cílového uzlu (v tomto případě uzlu x a v). Redukcí výše uvedené akce a dostaneme akci $a' = (z)[\ x(y)\ \bar{x}\langle z\rangle\ ]\langle yz\rangle$ (viz obrázek 4.3), nad níž můžeme provést další redukci. Výsledná akce má pak tvar $a'' = (z)\langle zz\rangle$ (viz obrázek 4.4).



Obrázek 4.3: π-net akce $a' = (z)[\ x(y)\ \bar{x}\langle z\rangle\ ]\langle yz\rangle$ — převzato z [12].



Obrázek 4.4: π-net akce $a'' = (z)\langle zz\rangle$ — převzato z [12].

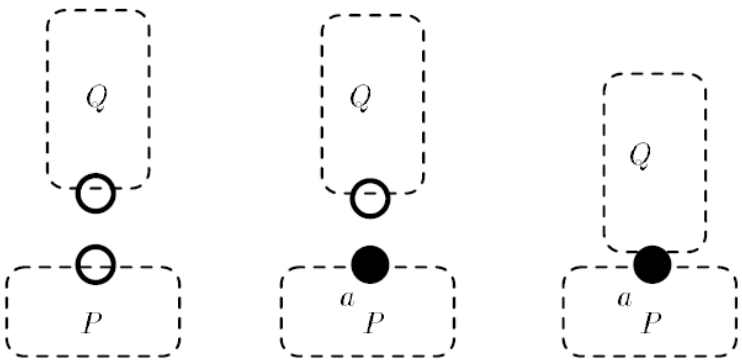
4.2.2 Diagramy interakce

Grafickou reprezentací, která se značně podobá π-nets v přístupu k redukcím (konkrétně myšlence sloučení uzlů), jsou diagramy interakce [14], jejichž autorem je Joachim Parrow (jeden z tvůrců π-kalkulu). Diagramy interakce byly navrženy pro grafický popis konkurentních procesů se zaměřením na vzájemnou interakci. Kruhové uzly grafu představují lokace neboli jména. Uzly mohou být propojeny vstupní nebo výstupní orientvanou hranou, která reprezentuje příslušnou akci vstupu či výstupu. Na obrázku 4.5 je znázorněna vstupní akce provedená prostřednictvím jména a. Obrázek 4.6 znázorňuje výstupní akci, kdy volné jméno b je komunikováno prostřednictvím jména a.

Obrázek 4.5: Grafické znázornění vstupní akce $a(x)$ — převzato z [14].

Obrázek 4.6: Grafické znázornění výstupní akce $\bar{a}\langle b\rangle$ — převzato z [14].

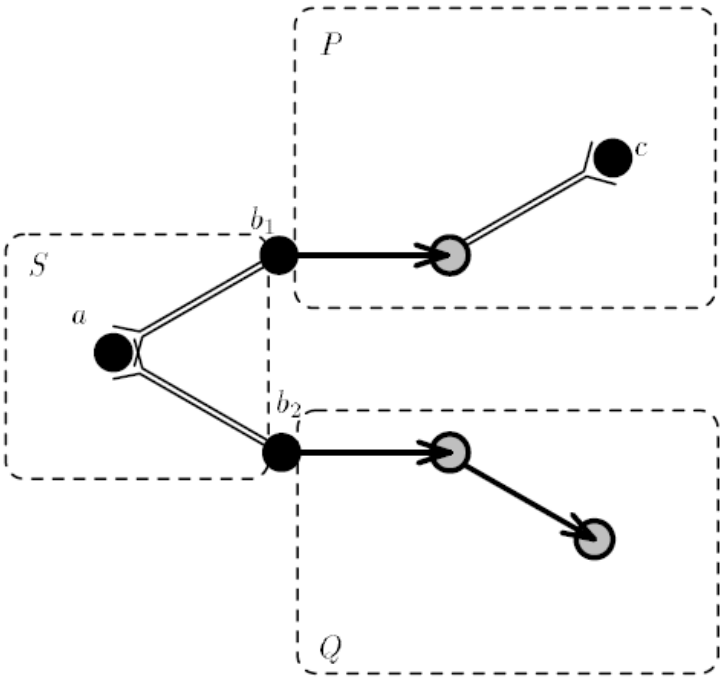
Výplň uzlu udává význam jména. Volná jména jsou znázorněna vyplněným (černým) kruhem s postranním označením. Nevyplněný (bílý) kruh pak představuje parametr neboli abstrakci jména. Příklad aplikace je znázorněn na obrázku 4.7 Iniciálně oddělené procesy P a Q (znázorněné čárkovaně) mají oba po jednom parametru. Vázané jméno a je nejprve aplikováno na parametr procesu P a následně na parametr procesu Q. Oba procesy tak aplikací získali přístup ke sdílenému jménu a.



Obrázek 4.7: Příklad postupné aplikace volného jména a na proces P a Q — převzato z [14].

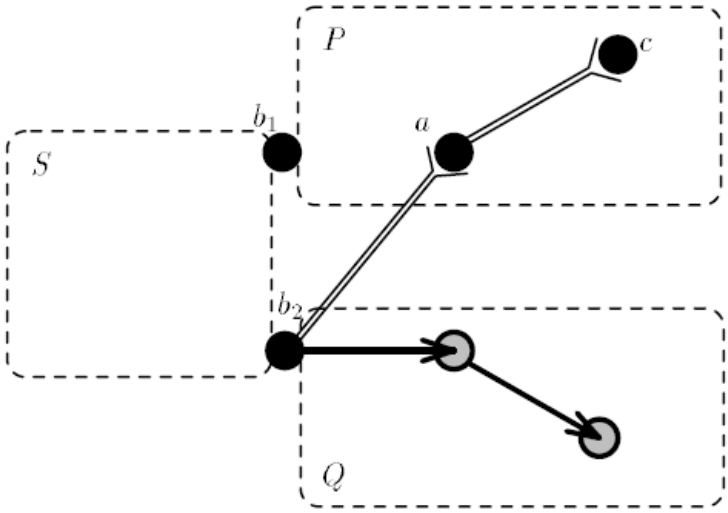
Jako speciální typ parametru můžeme chápat vstupní proměnnou (tj. cílový uzel vstupní akce), který se značí šedou výplní uzlu (viz obrázek 4.5).

Vstupní a výstupní akce provedená prostřednictvím stejného jména tvoří interakci. Obdobně jako u π-nets, tato interakce může být redukována odstraněním vstupní a výstupní hrany a sjednocením zdrojového a cílového portu (jména). Systém na obrázku 4.8 obsahuje dvě interakce (skrz b_1 a b_2).

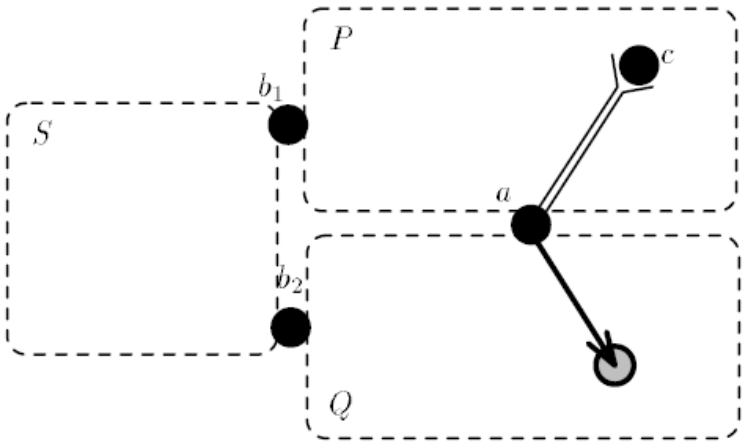


Obrázek 4.8: Grafické znázornění systému tvořeného procesy S, P a Q, kde S může komunikovat s P prostřednictvím portu b_1 a s Q prostřednictvím b_2 — převzato z [14].

Redukcí první z nich získá proces P přímý přístup ke jménu a (viz obrázek 4.9). Provedením druhé redukce získá přístup ke jménu a i proces Q (viz obrázek 4.10). Oba procesy tak po provedení obou redukcí sdílejí jméno a, pomocí něhož mohou komunikovat přímo, bez zásahu procesu S.

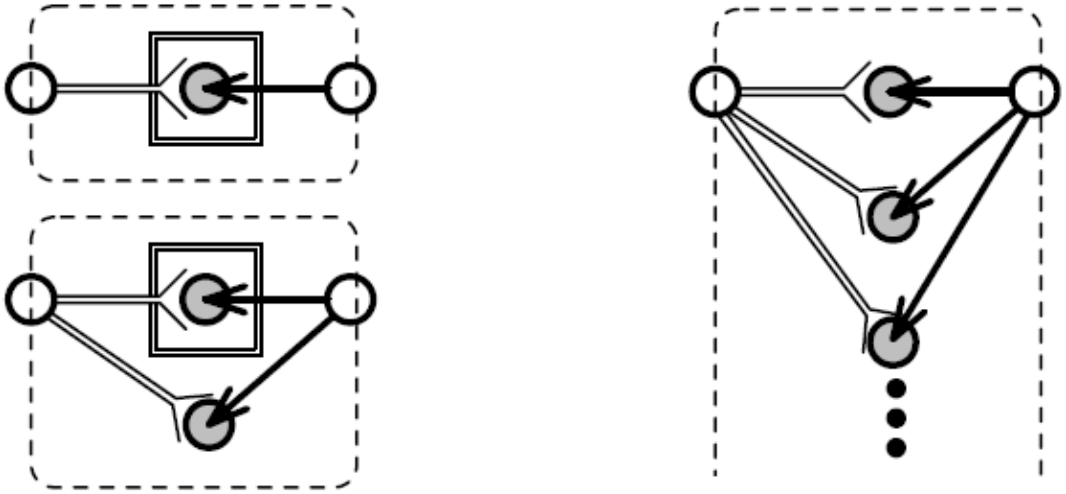


Obrázek 4.9: Grafické znázornění systému z obrázku 4.8 po provedení první redukce interakce (na b_1) — převzato z [14].

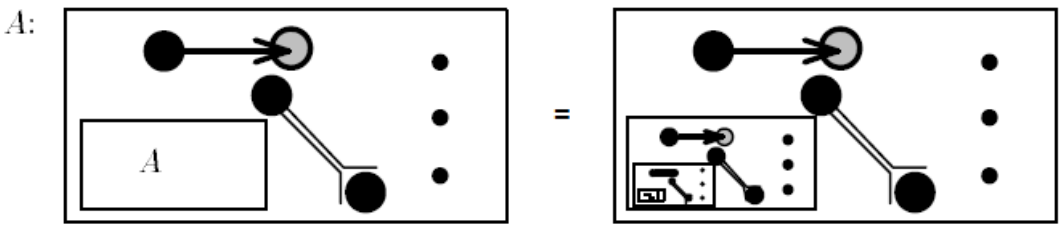


Obrázek 4.10: Grafické znázornění systému z obrázku 4.8 po provedení druhé redukce interakce (na b_2) — převzato z [14].

Diagramy interakce umožňují modelovat nekonečné chování pomocí rekurze, kdy část diagramu může být součástí sebe sama, a replikace, pomocí níž může být vytvořen neomezený počet kopií určité části diagramu. Příklad replikace znázorněné dvojitým rámečkem lze vidět na obrázku 4.11. Příklad rekurze je poté znázorněn na obrázku 4.12.



Obrázek 4.11: Příklad tří ekvivalentních reprezentací replikace. Systém na obrázku představuje neomezený buffer, který může opakovaně přijímat hodnoty ze vstupního portu a odesílat je na výstupní port — převzato z [14].



Obrázek 4.12: Grafické znázornění rekurze procesů — převzato z [14].

4.2.3 Přepisování grafu

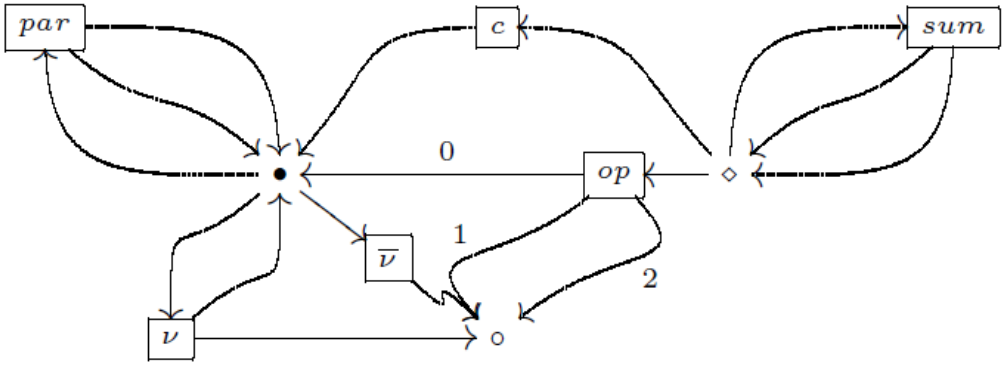
Dalším přístupem ke grafické interpretaci výrazů π-kalkulu je přepisování grafu. Tento přístup prezentoval ve svém článku [6] Fabio Gadducci v roce 2007.

Každý výraz π-kalkulu může být zakódován do podoby orientovaného grafu, jehož uzly reprezentují jména a procesy (podprocesy). Grafická podoba uzlu určuje o jaký typ entity se jedná. Jméno je značeno $\circ$, proces $\bullet$ a suma $\diamond$. Označené orientované hrany spojující jednotlivé uzly pak reprezentují operace. Každá hrana je označena příslušným názvem operace:

- sum - suma procesů
- par - paralelní kompozice procesů
- in - vstupní akce
- out - výstupní akce

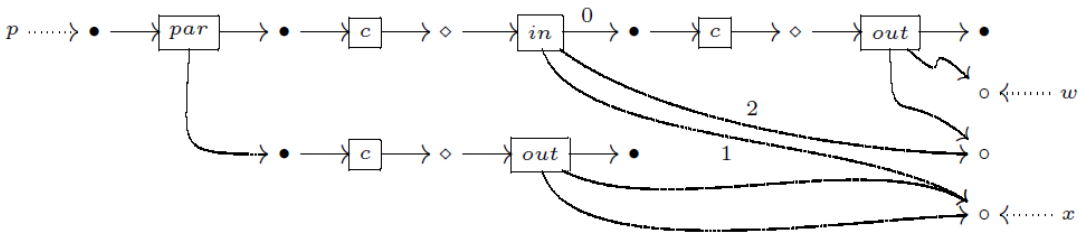
- ν - restrikce
- $\bar{\nu}$ - restrikce bez pokračování
- c - vynucení sumy uvnitř kontextu procesu

Každá hrana má nejvýše jeden zdrojový uzel a jeden či více cílových uzlů, které se mohou lišit svým typem. Například hrana představující operaci výstupní akce *out* má jeden zdrojový uzel $\diamond$ a 3 výstupní uzly $\langle \circ, \circ, \bullet \rangle$, z nichž první dva označují jména (jedno pro komunikační kanál a jedno pro přenášenou hodnotu) a poslední podproces, kterým se má pokračovat po vykonání výstupní akce. Graf typů pro jednotlivé operace ($op \in \{in, out\}$) je zobrazen na obrázku 4.13.



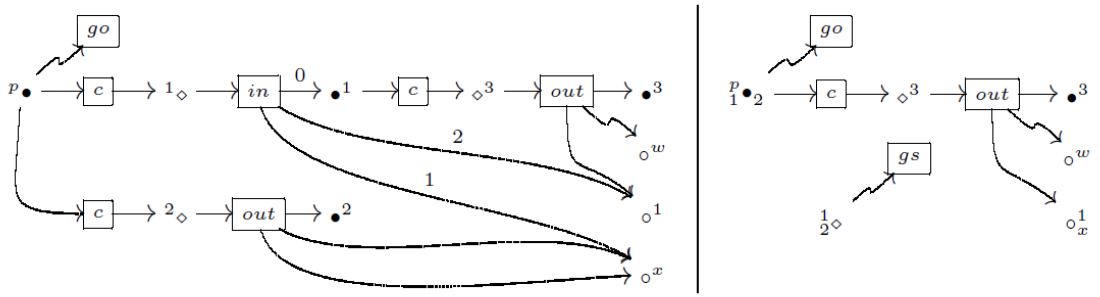
Obrázek 4.13: Graf typů pro jednotlivé operace, ($op \in \{in, out\}$) — převzato z [6].

Výraz $x(x).\bar{z}\langle w\rangle.0 \mid \bar{x}\langle x\rangle.0$ je možné graficky reprezentvat formou stromu uvedeného na obrázku 4.14.



Obrázek 4.14: Grafická reprezentace výrazu $x(x).\bar{z}\langle w\rangle.0 \mid \bar{x}\langle x\rangle.0$ — převzato z [6].

Jeden přepisovací krok pro výše uvedený výraz neboli simulace přechodu $x(x).\bar{z}\langle w\rangle.0 \mid \bar{x}\langle x\rangle.0 \rightarrow \bar{x}\langle w\rangle.0$ je znázorněna na obrázku 4.15. Hrany *go* a *gs* mají pomocný charakter (*go* označuje vstupní bod výpočtu a *gs* sumu procesů, která je pozůstatkem původního kódování grafu a bude po provedení redukce odstraněna).



Obrázek 4.15: Grafická reprezentace redukce $x(x).\bar{z}\langle w\rangle.0 \mid \bar{x}\langle x\rangle.0 \to \bar{x}\langle w\rangle.0$ — převzato z [6].

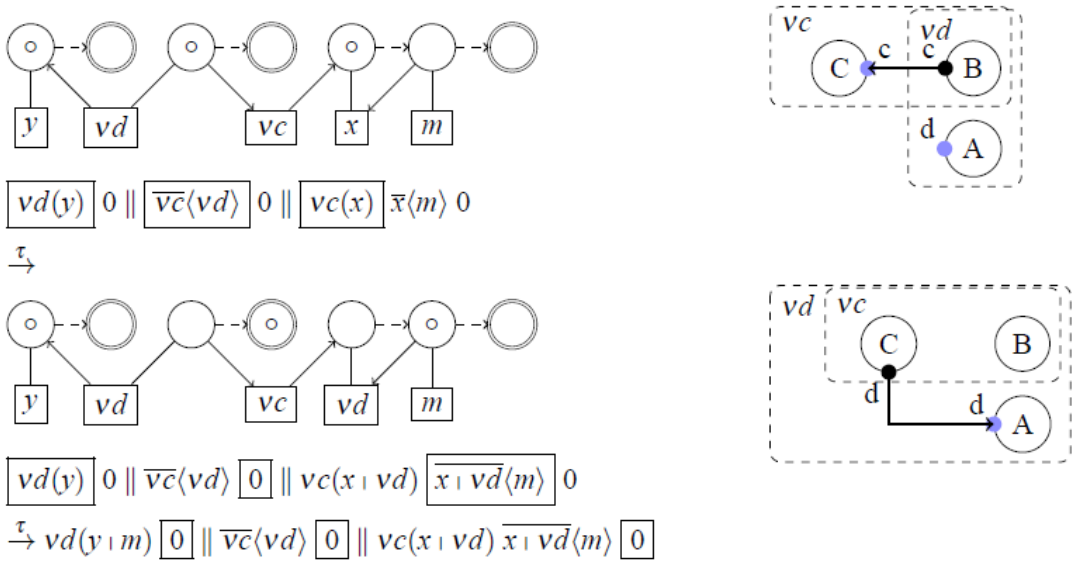
4.2.4 π-graphs

π-graphs je grafické paradigma volně inspirované Petriho sítěmi [16]. Jeho charakteristickou vlastností je statičnost. Zatímco ve všech výše uvedených grafických interpretacích docházelo při simulaci přechodu či redukce ke změně struktury grafu, tj. jednalo se o dynamické diagramy, hlavní myšlenkou π-graphs je pohybovat pouze jmény okolo statického grafu.

Každý proces je v π-graphs modelován jako neprázdná sekvence prefixů (značené kruhovými uzly) explicitně ukončená prázdným procesem 0 (značen uzlem s dvojitou čarou). Řídící tok je pak znázorněn prostřednictvím orientovaných čárkovaných hran spojujících jednotlivé uzly a pomocí známky (tokenu), která označuje aktuálně prováděný výpočetní krok a může tak být předávána z jednoho uzlu do druhého.

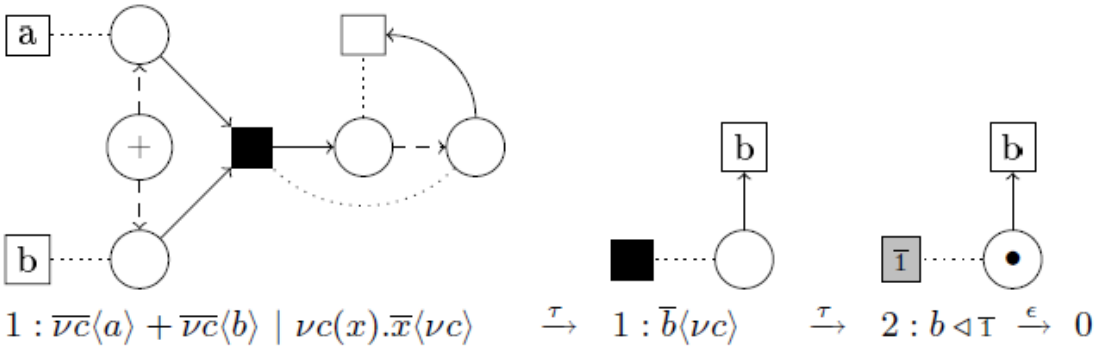
Datová část modelu (jména a kanály používané pro komunikaci) je realizována pomocí krabicových uzlů, které mohou být za běhu instanciovány konkrétním jménem. Po instanciaci je toto jméno vepsáno přímo do uzlu, zatímco původní formální jméno je zapsáno vedle něj. Každý z krabicových uzlů může být vázán na jeden nebo více kruhových uzlů, což znázorňuje uplatnění příslušného jména v prefixu. Nepřerušovaná orientovaná hrana směřující od prefixu ke jménu značí, že dané jméno má funkci výstupního kanálu, v opačném případě vstupního kanálu. Neorientované (zpravidla tečkované) hrany značí referenci na přenášená jména.

Na obrázku 4.16 je znázorněn jeden redukční krok procesu tvořeného paralelní kompozicí tří vzájemně interagujících podprocesů A, B a C. Procesy B a C sdílejí soukromý kanál c pomocí něhož proces B předá procesu C referenci na soukromé jméno d, které bylo původně přístupné pouze procesu A a B.



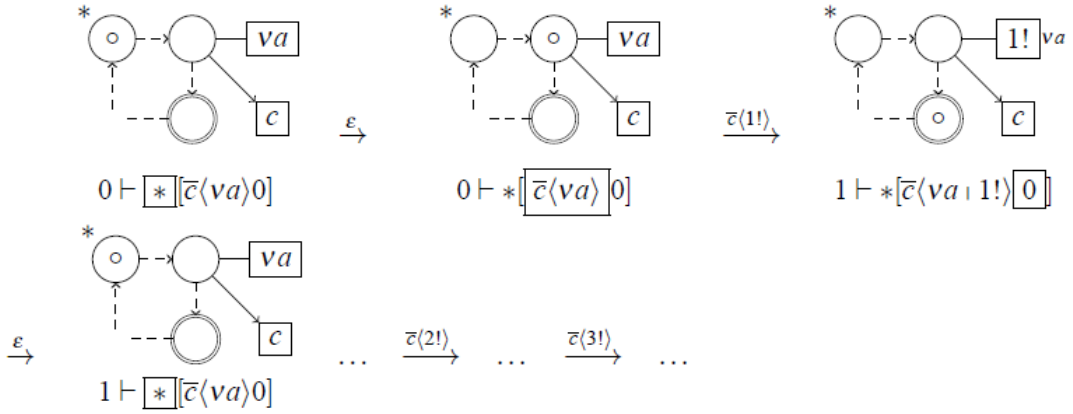
Obrázek 4.16: Grafické znázornění jednoho výpočetního kroku systému tvořeného třemi procesy A, B a C. Současný stav každého procesu je v π-graph značen pomocí známky uvnitř kruhového uzlu. V textové reprezentaci pod ním je pak prováděná část výpočtu vyznačena rámečkem. V pravo jsou uvedeny odpovídající diagramy propojení znázorňující strukturu systému — převzato z [16].

π-graphs využívá princip Lamportových logických hodin pro generování nových jedinečných jmen, která jsou různá od všech ostatních jmen použitých v systému. Takové jméno musí být vytvořeno vždy, když dochází k odeslání soukromého jména prostřednictvím veřejného kanálu. Příklad takovéto komunikace uvnitř procesu $(\nu c)((\bar{c}\langle a\rangle + \bar{c}\langle b\rangle) \mid c(x).\bar{x}\langle c\rangle)$ je znázorněn na obrázku 4.17, kdy je nejprve veřejné jméno b odesláno prostřednictvím soukromého kanálu c (označen černě) a následně soukromé jméno c odesláno prostřednictvím veřejného jména b. V dalším kroku redukce je tak c nahrazeno za vygenerované jedinečné jméno $\bar{1}$.



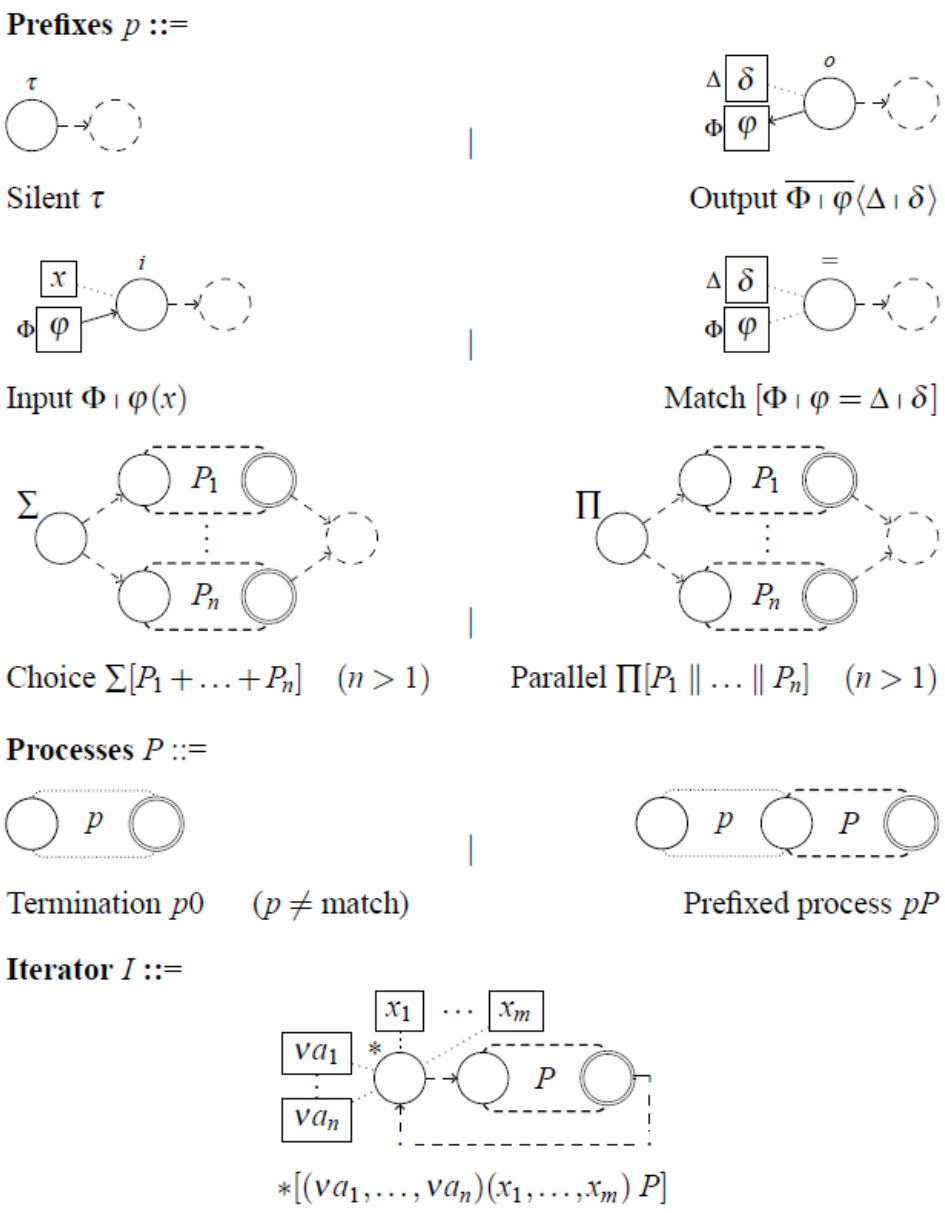
Obrázek 4.17: Grafické znázornění dvou redukčních kroků procesu $(\nu c)((\bar{c}\langle a\rangle + \bar{c}\langle b\rangle) \mid c(x).\bar{x}\langle c\rangle)$ včetně vygenerování jedinečného jména — převzato z [15].

Pro modelování nekonečného chování jsou používány takzvané iterátory. Na obrázku 4.18 je možné vidět příklad použití iterátoru společně s generováním jedinečných jmen.



Obrázek 4.18: Příklad grafického znázornění cyklu iterátoru s generováním jedinečných jmen — převzato z [16].

Syntax jazyka π-graphs, tedy přehled grafického znázornění prefixů a dalších konstrukcí je uveden na obrázku 4.19.



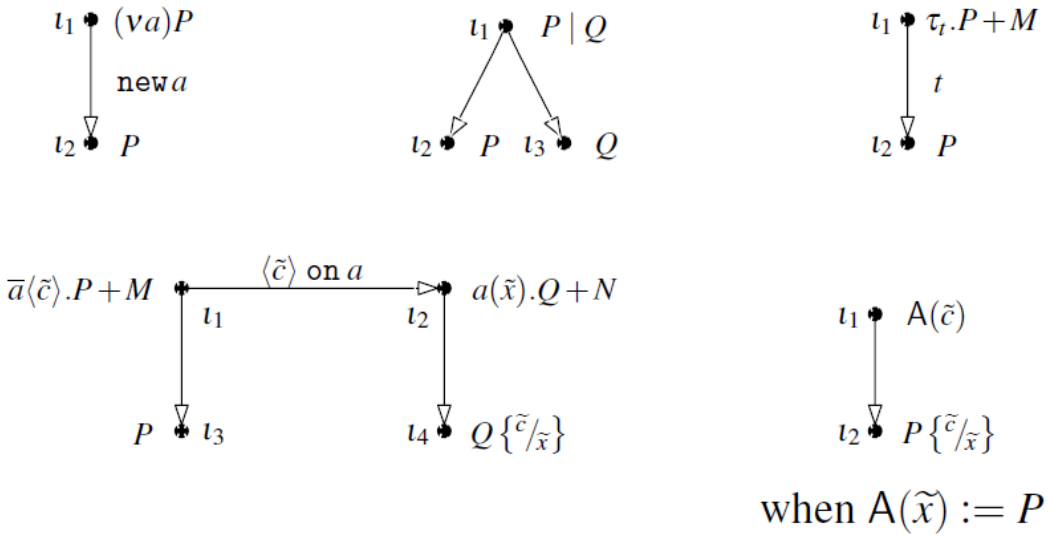
Obrázek 4.19: Základní konstrukce grafického modelu π-nets — převzato z [16].

4.2.5 Pi-charts

Pi-chart je orientovaný acyklický graf navržený pro grafické vyjádření sémantiky π-kalkulu jakožto alternativy k relaci redukce a přechodu [4]. Prcesy jsou znázorněny jako uzly grafu, přičemž každá hrana reprezentuje buď výpočetní krok procesu, nebo interakci mezi procesy prostřednictvím zasílání zpráv. Obdobně jako v diagramech MSC, vertikální hrany reprezentují řídící tok a horizontální hrany datový tok. Každý pi-chart tak reprezentuje výpočet,

který začíná procesy (uzly), které nemají žádného předchůdce a postupuje směrem dolů až k procesům (uzlům), které nemají žádného následníka.

Na obrázku 4.20 je uveden přehled všech primitivních grafů, které odpovídají základním konstrukcím jazyka π-kalkulu. Jejich spojením je pak možné sestavit pi-chart odpovídající komplexnějším procesním výrazům.

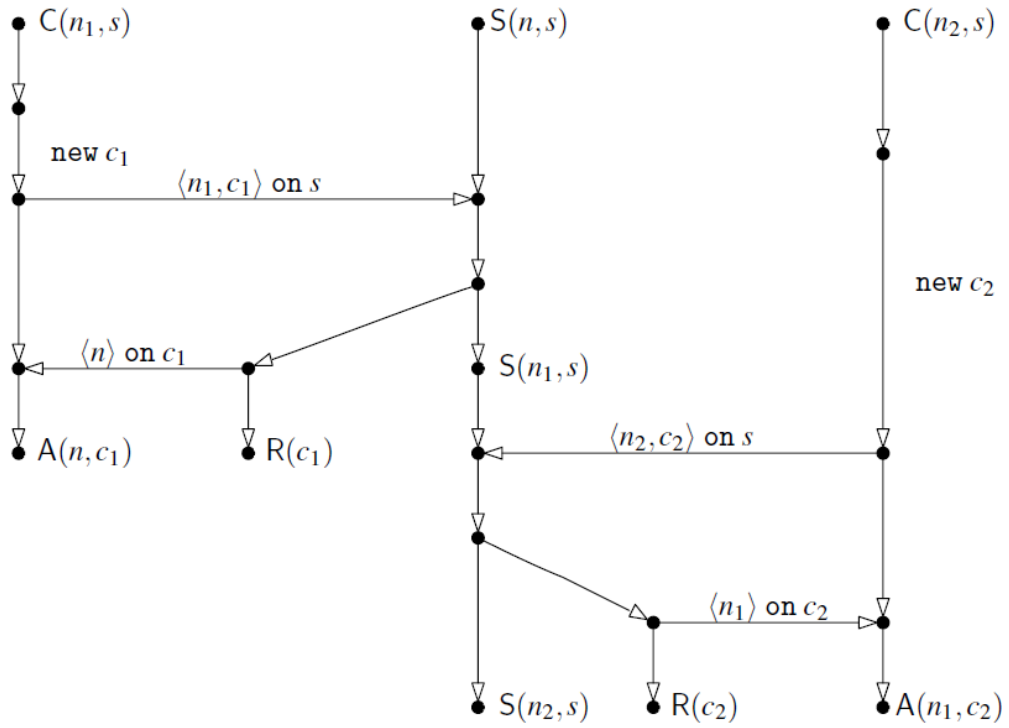


Obrázek 4.20: Základní konstrukce grafického modelu pi-chart — převzato z [4].

Definujme nyní prostřednictvím výrazů π-kalkulu proces klienta C a proces serveru S následovně:

$$S(n,s) \stackrel{\text{def}}{=} s(n',c).(\bar{c}\langle n\rangle.R(c) \mid S(n',s))$$
$$C(n',s) \stackrel{\text{def}}{=} (\nu c)\bar{s}\langle n',c\rangle.c(n).A(n,c)$$

Pi-chart na obrázku 4.21 znázorňuje vzájemné interakce mezi jedním serverem a dvěma klienty.



Obrázek 4.21: Grafické znázornění vývoje systému tvořeného serverem S a dvěma klienty C — převzato z [4].

Kapitola 5

Analýza a návrh nástroje pro vizualizaci výrazů π-kalkulu

Přestože π-kalkul není ve své podstatě nijak zvlášť složitý, abstraktní matematická forma, kterou je prezentován ve většině publikací, nemusí být pro každého na první pohled snadno srozumitelná. Nástroj, který by umožňoval výrazy π-kalkulu vizualizovat by tak mohl být přínosem jednak pro ty, kteří se s algebrou π-kalkulu teprve seznamují (jakožto pomůcka při studiu), tak i pro návrháře mobilních systémů (jakožto nástroj k ověření správnosti výrazů). V případě složitějších systémů totiž může být textová reprezentace výrazů π-kalkulu značně nepřehledná a tím pádem i náchylná k chybám. Například nesprávným umístěním jediné závorky může dojít k zásadní změně chování modelovaného systému. Rovněž hledání všech možných redukcí v textové reprezentaci komplexnějšího výrazu může být poměrně časově náročné.

Je obecně uznávaným faktem, že grafický popis, v porovnání s popisem textovým, obvykle poskytuje rychlejší vhled do určité problematiky [5], přičemž vizualizace je pak zvláště přínosná, pokud se jedná o znázornění vztahů mezi komponentami a jejich vzájemné komunikace, jako je tomu právě i v případě π-kalkulu.

Z průzkumu existujících aplikací, jehož výsledky byly popsány v předchozí kapitole, vyplynulo, že dosud nebyl vytvořen žádnný nástroj, jenž by pracoval s grafickou reprezentací výrazů π-kalkulu. Návrh a následná implementace nástroje, který by umožňoval převod textové reprezentace výrazů π-kalkulu do grafické podoby , a tím poskytoval uživatelům výše zmíněné výhody, byly hlavním cílem této diplomové práce.

V této kapitole se budeme zabývat nejprve obecnou analýzou požadavků na funkčnost a podobu vytvořené aplikace, následně pak návrhem textové, grafické a interní reprezentace výrazů π-kalkulu. Na závěr popíšeme architekturu vytvořené aplikace a specifikujeme funkci jednotlivých komponent.

5.1 Analýza požadavků

Před samotným návrhem aplikace bylo potřeba rámcově specifikovat, jaké základní funkce a prostředky pro jejich provádění má aplikace uživateli nabízet. Pevně stanoveným požadavkem vycházejícím ze zadání byl převod výrazů z textové reprezentace do grafické s případným rozšířením o převod opačný. Z toho vyplývá nutná existence prostředků pro textový vstup a grafický výstup, případně pak i grafický vstup a textový výstup. Součástí

aplikace tak zřejmě bude grafické uživatelské rozhraní, umožňující uživatelům pomocí svých komponent vkládat či prohlížet textovou i grafickou reprezentací výrazů.

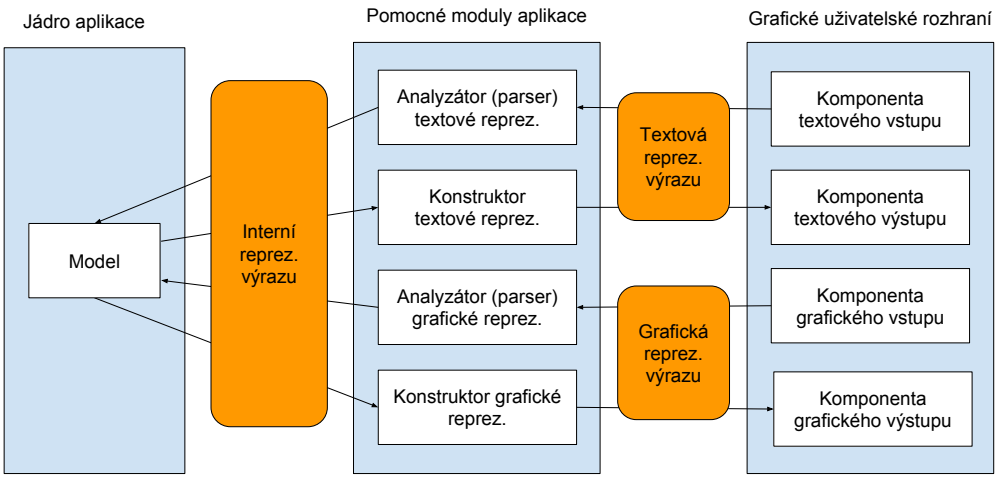
Za účelem podpory modularity, principu znovupoužití a modelování nekonečného chování, bude uživateli umožněno prostřednictvím textového vstupu specifikovat nejen konkrétní výrazy určené k vizualizaci, ale také definovat parametrizované procesy. Aplikace těchto procesů pak budou moci být použity jako součást libovolného výrazu či jiné definice procesu. Aplikace by pak zřejmě měla nabízet možnost tyto definované procesy prohlížet, upravovat, odstraňovat a případně také uložit a opětovně načíst do/z textového souboru.

Uživateli bude dále umožněno provádět nad specifikovaným výrazem redukce a dle potřeby případné další modifikace (např. expanzi replikace). Aplikace v takovém případě zajistí, že uživateli bude vždy prezentována aktuální textová i grafická podoba zpracovávaného výrazu.

Před provedením redukce uživatel označí prefixy, které budou vstupem redukce, a to buď označením v grafické reprezentaci výrazu nebo výběrem ze seznamu možných redukcí. Při výběru pouze vstupního či výstupního prefixu budou uživateli nabídnuty komplentární prefixy.

Celkový stav aplikace bude možné uložit a opětovně načíst do/ze souboru. Grafický výstup pak bude možné exportovat do některého z dostupných grafických formátů, za účelem snadného vložení do dokumentace či technické zprávy. Grafické uživatelské rozhraní bude mimo komponenty pro textový a grafický vstup a výstup obsahovat také kontrolní prvky, které uživateli zpřístupní výše popsané funkce aplikace.

Obrázek 5.1 ve zjednodušené formě znázorňuje transformace prováděné mezi textovou, grafickou a interní reprezentací výrazů k nimž dochází uvnitř aplikace. Uživatel nejprve prostřednictvím uživatelského rozhraní specifikuje textovou nebo grafickou reprezentaci výrazu. Ta je pomocí příslušných modulů převedena na interní reprezentaci, jež tvoří součást aplikačního modelu. Uložená interní reprezentace může být následně dále zpracovávána a modifikována na základě příkazů zadaných uživatelem (např. příkazem k provedení redukce), přičemž reakcí na jakoukoliv změnu interní reprezentace je vyvolání zpětné transformace na textovou a grafickou reprezentaci, která je prostřednictvím uživatelského rozhraní prezentována uživateli.



Obrázek 5.1: Schéma transformací mezi textovou, grafickou a interní reprezentací výrazů.

5.2 Textová reprezentace výrazů

Za účelem zachování vzájemné kompatibility byl při návrhu textové ASCII reprezentace výrazů π-kalkulu brán ohled na již existující aplikace, zejména pak nástroj MWB a ABC. Finální podoba navržené textové reprezentace je popsána následující gramatikou:

`agent` $Id\ Params$ `=` $Proc$

$$
\begin{array}{rcl}
Params & := & \langle \text{empty} \rangle \\
 & & \texttt{(}\ nlist\ \texttt{)}
\end{array}
$$

$$
\begin{array}{rcl}
Oparams & := & \langle \text{empty} \rangle \\
 & & \texttt{<}\ nlist\ \texttt{>}
\end{array}
$$

$$
\begin{array}{rcl}
nlist & := & name \\
 & & name\ \texttt{,}\ nlist
\end{array}
$$

$$
name := \texttt{[a-z][a-zA-Z0-9\_-]*}
$$

$$
Id := \texttt{[A-Z][a-zA-Z0-9\_-]*}
$$

$$
\begin{array}{rcl}
Proc & := & \texttt{0} \\
 & & \pi\texttt{.}Proc \\
 & & \texttt{[}name\ \texttt{=}\ name\texttt{]}Proc \\
 & & Proc\ \texttt{|}\ Proc \\
 & & Proc\ \texttt{+}\ Proc \\
 & & Id\ Oparams \\
 & & Id\ Params \\
 & & \texttt{(}\ \texttt{\^{}}\ nlist\ \texttt{)}\ Proc \\
 & & \texttt{(}\ \texttt{\textbackslash}\ nlist\ \texttt{)}\ Proc \\
 & & \texttt{[}\ nlist\ \texttt{]}\ Proc \\
 & & \texttt{(}\ Proc\ \texttt{)}
\end{array}
$$

$$
\begin{array}{rcl}
\pi & := & name\ Params \\
 & & \texttt{'}name\ Oparams \\
 & & \texttt{t}
\end{array}
$$

5.3 Grafická reprezentace výrazů

Použitá grafická reprezentace byla inspirována vizuálními modely popsanými dříve, konkrétně pak modelem π-graphs, co se týče vizualizace jmen a operací π-kalkulu jako uzlů rozdílných typů. Výraz znázorněný pomocí navržené grafické reprezentace má podobu orientovaného grafu jehož uzly reprezentují operace (suma, paralelní kompoice, prefix, atp.), aplikace parametrizovaných procesů a jména.

Operace π-kalkulu jsou znázorněny kruhovým uzlem bílé barvy označeným symbolem odpovídající operace. Tabulka 5.1 definuje mapování operací na symboly příslušných uzlů.

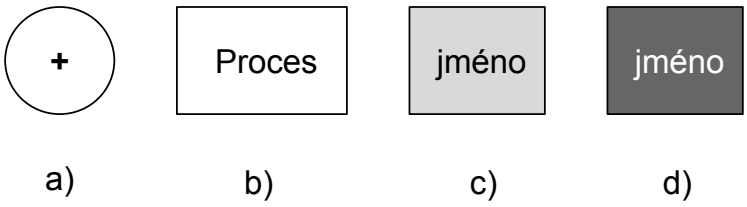
prázdná suma	0
suma	+
paralelní kompozice	|
replikace	!
vstupní prefix	i
výstupní prefix	o
nepozorovatelný prefix	t
prefix shody	=

Tabulka 5.1: Mapování operací π-kalkulu na symboly uvedené v grafické reprezentaci uzlu.

Restrikce, jako operace, která neovlivňuje strukturu výrazu, ale pouze typ použitých jmen, není v grafické reprezentaci znázorněna samostatným uzlem. Namísto toho je reflektována vizuální podobou uzlů reprezentující jména.

Jména jsou v grafické reprezentaci znázorněna obdélným uzlem šedé barvy označeným názvem daného jména. Uzly reprezentující privátní jména (tj. jména vytvořená operací restrikce) jsou vybarvena tmavším odstínem šedé.

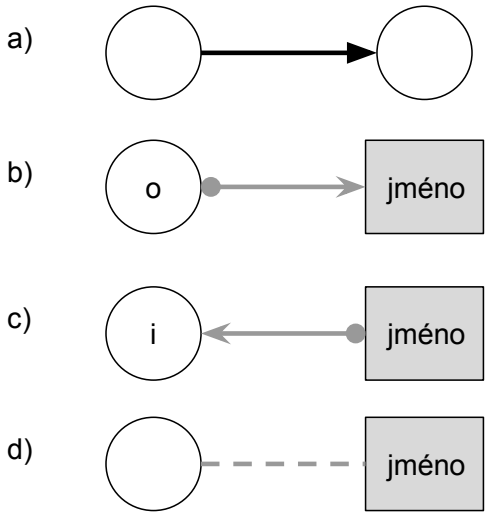
Aplikace procesů jsou znázorněny obdélným uzlem bílé barvy označeným identifikátorem daného procesu. Na obrázku 5.2 je možné vidět všechny výše popsané typy uzlů.



Obrázek 5.2: Typy uzlů v navržené grafické reprezentaci. (a) operace, (b) aplikace procesu, (c) jméno, (d) privátní jméno.

Hrany se v použité grafické reprezentaci liší svým vzhledem podle toho, jaké typy uzlů spojují. Na obrázku 5.3 je zobrazen přehled všech typů hran. První z nich je orientovaná hrana spojující dva uzly operací, případně pak uzel operace s uzlem procesu. Orientace hrany modeluje vztah [výraz $\rightarrow$ podvýraz], kdy podgraf grafu, jehož kořenem je cílový uzel, reprezentuje podvýraz výrazu reprezentovaného podgrafem grafu, jehož kořenem je zdrojový uzel.

Hrany spojující jména s operacemi či procesy jsou znázorněny šedou barvou. V případě, že spojené jméno reprezentuje komunikační kanál vstupně-výstupní akce (tj. váže se k operaci vstupního nebo výstupního prefixu), hrana je orientovaná podle směru prováděné akce – tj. u výstupní akce směrem od uzlu operace ke jménu a u vstupní akce od jména směrem k uzlu operace. Tyto hrany mají navíc na obou koncích specifické ukončení, což usnadňuje vyhledávání možných redukcí. Všechny ostatní hrany, které připojují k operaci či procesu jména reprezentující parametry, nejsou orientované a jsou znázorněny přerušovanou čarou.

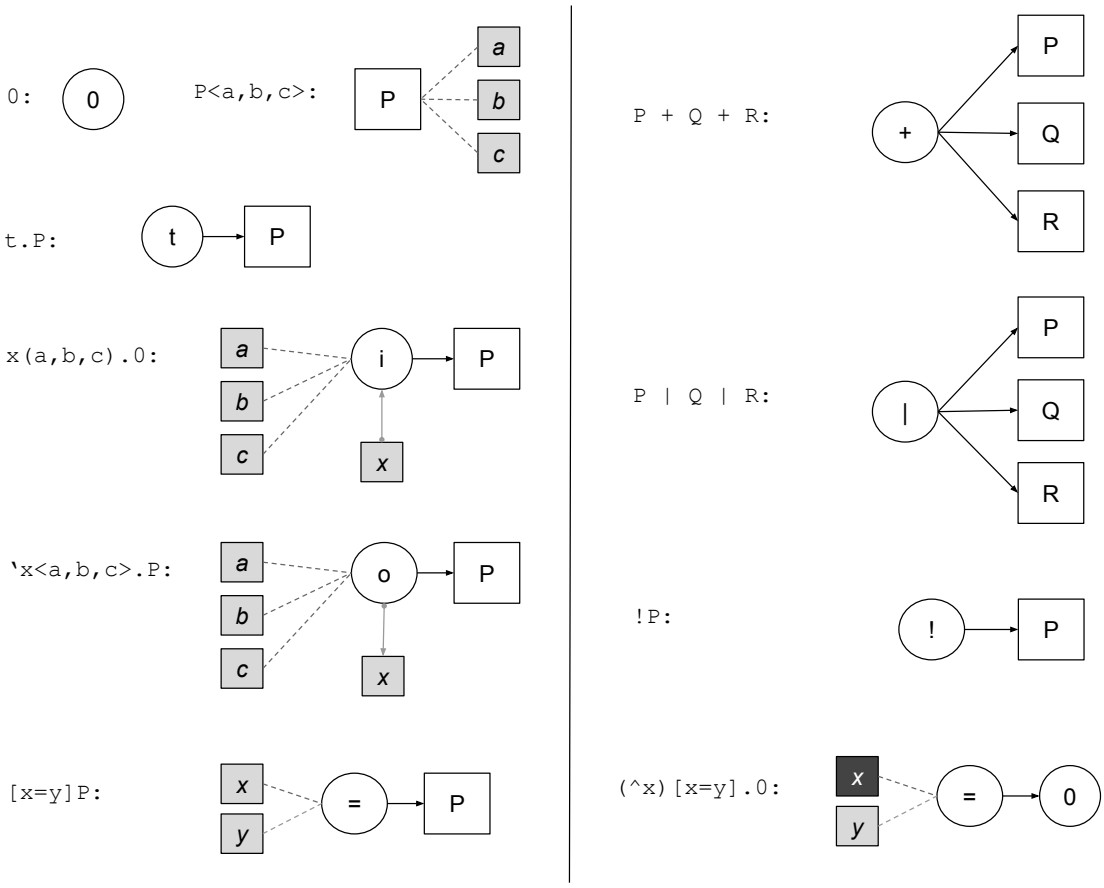


Obrázek 5.3: Typy hran v navržené grafické reprezentaci.

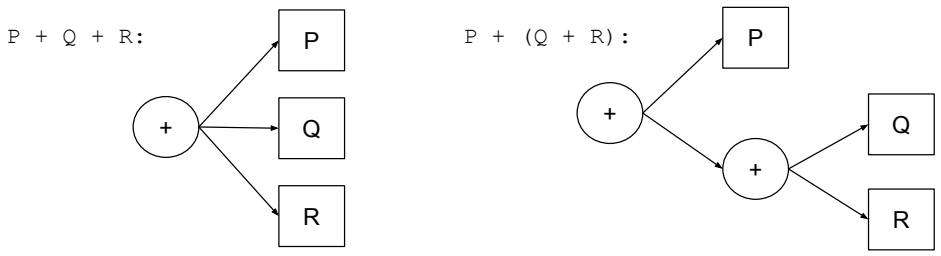
Na rozdíl od π-graphs je navržená grafická reprezentace dynamická – provedení redukce mění strukturu grafu. Některé uzly či celé podgrafy jsou redukcí odstraněny a hrany vedoucí k uzlům reprezentujícím substituovaná jména jsou přesměrovány. Důsledkem této modifikace je pak každé jméno reprezentováno právě jedním uzlem, což umožňuje snadnější určení rozsahu jmen a potenciálních redukcí. Rozsah jména je určen všemi hranami vycházejícími z konkrétního jména. Případná redukce má pak podobu vstupního a výstupního uzlu propojených příslušným typem vstupně-výstupní hrany ke stejnému jménu. Graf navíc po provedení redukce znázorňuje vždy pouze výpočetně proveditelnou část výrazu a není tedy nutné uzly aktuálně prováděných výpočetních kroků označovat pomocí známky (tokenu), jako tomu bylo v modelu π-graphs.

Obrázek 5.4 obsahuje přehled elementárních konstrukcí výrazů procesní algebry π-kalkul zakreslených pomocí navržené grafické reprezentace. Použití závorek v textové reprezentaci výrazu je v grafické reprezentaci reflektováno topologií grafu. Obrázek 5.5 demonstruje rozdíl mezi grafickou reprezentací výrazu $P + Q + R$ a výrazu $P + (Q + R)$.

Aplikace definovaného procesu, neboli též konkretizace, je speciálním případem výrazu, neboť ji lze vnímat jak ve smyslu abstrakce daného procesu, tak i jako plnohodnotný podvýraz určený definicí procesu. Zatímco v prvním případě je aplikace procesu znázorněna jediným uzlem označeným názvem daného procesu, v druhém případě se pak jedná o podgraf reprezentující daný podvýraz. Vzhledem k tomu, že vytvořená aplikace umožňuje interakci uživatele, může si uživatel u každé použité aplikace procesu zvolit požadovanou formu grafické reprezentace pomocí kontrolního tlačítka. Obrázek 5.6 znázorňuje tři možné varianty



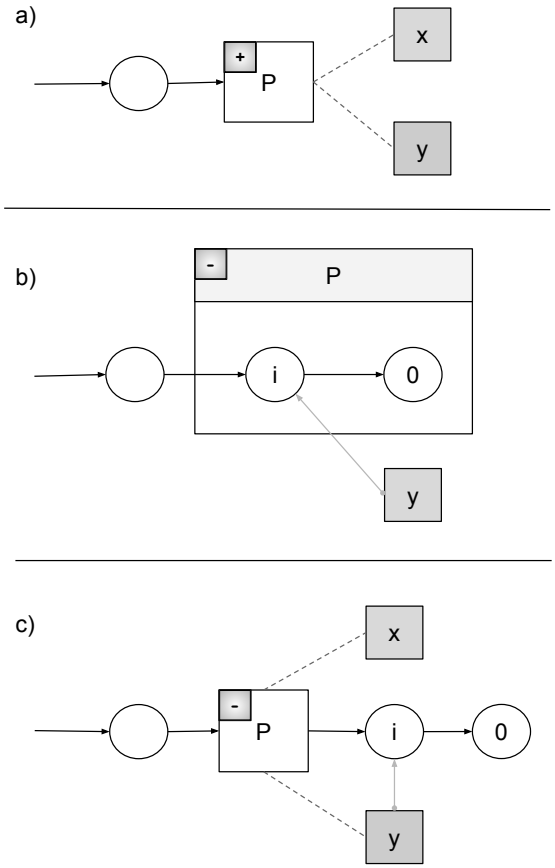
Obrázek 5.4: Grafická reprezentace elementárních výrazů.



Obrázek 5.5: Rozdíl grafické reprezentace výrazu $P + Q + R$ a výrazu $P + (Q + R)$.

grafické reprezentace aplikace procesu P, jenž je definován jako proces se dvěma parametry a a b následovně: $P(a, b) = a.0$. Za parametry procesu jsou ve znázorněném příkladu dosazeny jména x a y. Jméno x dosazené za první parametr a, je v podvýrazu použito jako komunikační kanál vstupního prefixu, zatímco jméno y, které je aplikováno na parametr b, se v podvýrazu neuplatní.

První varianta (a) na obrázku 5.6 znázorňuje případ, kdy je vnitřní reprezentace procesu P abstrahována. Druhá varianta (b) znázorňuje podvýraz procesu P připojený k původnímu výrazu s použitím hierarchického uspořádání. V hierarchickém uspořádání jsou uzly podgrafu ohraničeny obalujícím uzlem konkrétního procesu (Pozn.: Tato varianta je podobná grafické reprezentaci používané diagramy interakce.) Poslední varianta (c) znázorňuje podvýraz procesu připojený k původnímu výrazu s použitím lineárního uspořádání. V případě lineárního uspořádání je podgraf podvýrazu navázán za příslušný uzel procesu.



Obrázek 5.6: Grafická reprezentace aplikace procesu.

5.4 Interní reprezentace výrazů

Navržená interní reprezentace výrazů je obecným formátem pro ukládání výrazů π-kalkulu uvnitř aplikace, který uchovává informace nezbytné pro strojovou analýzů a zpracování výrazů.

5.4.1 Reprezentace operací a procesů

Výrazy π-kalkulu jsou v interní reprezentaci organizovány do struktury obousměrně vázaného stromu, jehož uzly reprezentují operace π-kalkulu, abstrakce a konkretizace procesů nebo kořen výrazu. Tabulka 5.2 uvádí výčet všech typů uzlů.

Téměř každý uzel obsahuje odkaz na svého předchůdce a na jednoho či více následníků. Výjimku tvoří uzel Nil reprezentující prázdnou sumu, který nemá žádného následníka, a kořenový uzel, který nemá předchůdce. Speciálním případem je pak také abstrakce a konkretizace procesu.

Abstrakce procesu představuje kořenový uzel výrazu, jenž společně s názvem procesu tvoří definici daného procesu. Abstrakce má předchůdce pouze tehdy, pokud je navázána na odpovídající uzel konkretizace. Obdobně konkretizace má následníka pouze tehdy, je-li příslušný proces definován, tzn. existuje abstrakce, kterou by bylo možné navázat jako následníka a provedení tohoto navázání (instanciace) bylo explicitně vyžádáno.

Dalším uzlem, který má specifický vztah k jinému typu uzlu je replikace. Operace replikace je v navržené reprezentaci modelována pomocí dvou uzlů – speciálním typem paralelní kompozice a navazujícím uzlem replikace. Podstrom s kořenovým uzlem replikace přitom může reprezentovat buď původní replikační výraz nebo jeho kopii (tj. výraz vytvořený rozbalením replikace). V obou případech je však předchůdcem uzel paralelní kompozice vyhrazený pro replikace. Pro algoritmus vyhledávánní redukcí, popsaný v následující kapitole, je podstatné, aby následníkem dané paralelní kompozice byl vždy právě jeden originál a alespoň jedna kopie.

Součástí některých uzlů jsou také odkazy na použitá jména. Například uzel vstupního prefixu obsahuje jak odkaz na komunikační kanál, tak i seznam parametrů.

5.4.2 Reprezentace jmen

Způsob reprezentace jmen byl klíčovým problémem při návrhu interní reprezentace výrazů. V následujícím textu budou nejprve popsány dva nevyhovující způsoby vedoucí však k finálnímu řešení.

Reprezentace názvem

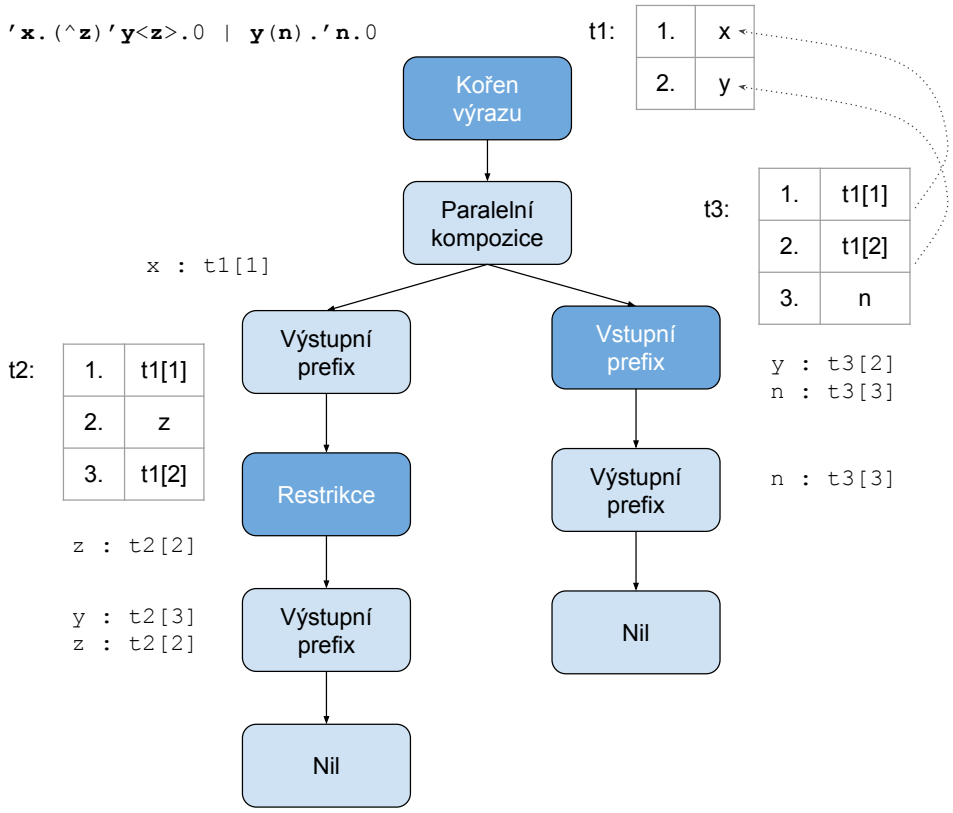
Jednou z možností by bylo reprezentovat jména uložená v uzlech pouze jejich názvem (tj. řetězcovým literálem). Tento naivní přístup je však nepoužitelný vzhledem k chybějící informaci o rozsahu jména. Výraz π-kalkulu totiž může obecně obsahovat několik jmen se stejným názvem, které se však liší svým rozsahem. Vezměme v úvahu výraz $(\nu a)a.0 \mid \bar{a}.0$. Tento výraz obsahuje dvě jména označená písmenem a, přičemž každé z nich má jiný rozsah – první z nich je privátní a jeho rozsah je určen podvýrazem $(\nu a)a.0$, zatímco rozsah druhého je určen podvýrazem $\bar{a}.0$. Dvě jména reprezentovaná stejným řetězcovým literálem, vyskytující se však v různých uzlech výrazu by tak nebylo možné objektivně porovnat na shodu, což by mělo negativní dopad na funkcionalitu aplikace (např. hledání potenciálních redukcí).

Reprezentace pomocí definičních tabulek

Možným řešením problému s rozsahem jmen je zavedení tabulek definovaných jmen. Tabulka definovaných jmen by byla součástí uzlů, které zavádějí nová jména a umožňují jejich volný výskyt v následném podstromu. Takovými uzly jsou konkrétně uzel vstupního prefixu, uzel

Typ uzlu	Přechůdce	Počet následníků	Jména
Nil (prázdná suma)	ano	0	ne
Nepozorovatelný prefix	ano	1	ne
Vstupní prefix	ano	1	komunikační kanál + seznam parametrů
Výstupní prefix	ano	1	komunikační kanál + seznam parametrů
Prefix shody	ano	1	dvě porovnávaná jména
Restrikce	ano	1	seznam omezených jmen
Suma	ano	>1	ne
Paralelní kompozice	ano	>1	ne
Paralel. kompozice (repl.)	ano	>1	ne
Replikace	ano (paralelní. komp.)	1	ne
Konkretizace procesu	ano	0/1 (abstrakce)	seznam argumentů
Abstrakce procesu	ano/ne (konkretizace)	1	seznam parametrů
Kořen výrazu	ne	1	seznam volných jmen

Tabulka 5.2: Přehled uzlů použitých v interní reprezentaci výrazů algebry π-kalkul.



Obrázek 5.7: Interní reprezentace výrazu $\bar{x}(\nu z)\bar{y}\langle z\rangle.0 \mid y(n).\bar{n}.0$ s využitím tabulek definovaných jmen.

restrikce, uzel abstrakce procesu a kořenový uzel. Ostatní uzly by pak obsahovaly místo názvu jména index do nejbližší tabulky definovaných jmen uložené v některém z předchůdců. Každá tabulka jmen by přitom obsahovala jak definice nových jmen, tak i reference na jména definovaná v některé z předchozích tabulek.

Obrázek 5.7 znázorňuje interní reprezentaci výrazu $\bar{x}(\nu z)\bar{y}\langle z\rangle.0 \mid y(n).\bar{n}.0$ s použitím tabulek definovaných jmen.

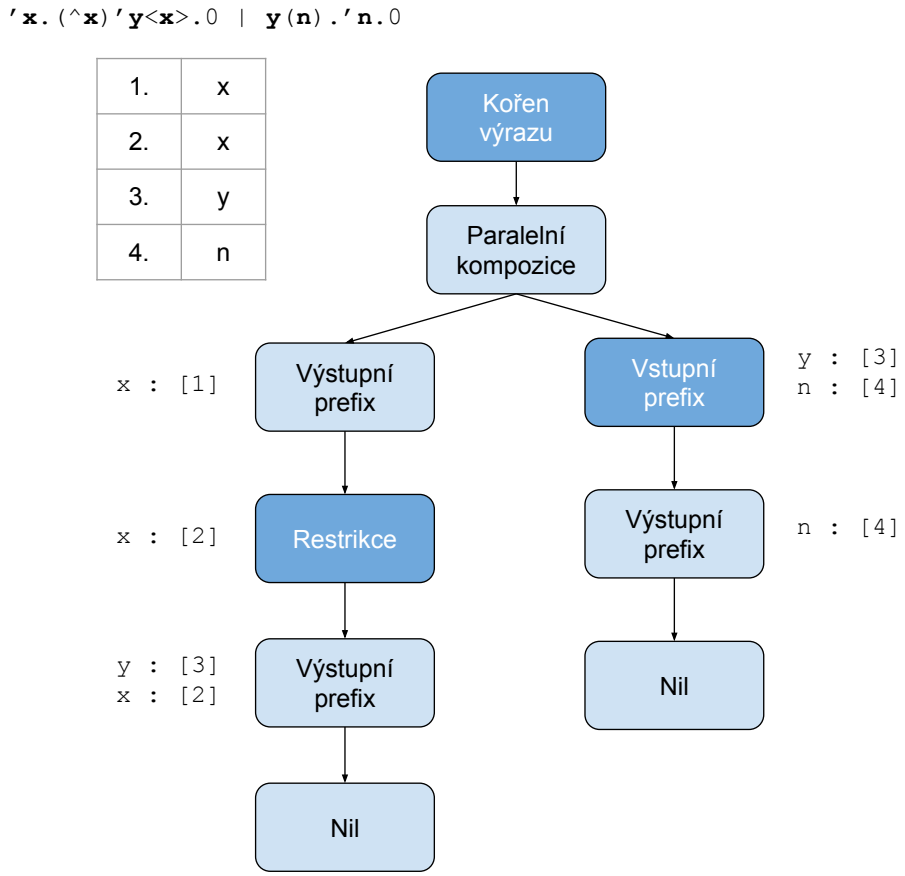
V této reprezentaci je rozsah jména, jehož reference je použita v libovolném uzlu, určen podstromem, jehož kořenový uzel obsahuje tabulku jmen s definicí daného jména. Při porovnávání dvou jmen na shodu je tak možné postupovat strukturou stromu směrem ke kořenovému uzlu a cestou přemapovávat reference vybraných jmen dle navšívených tabulek až k tabulce, která obsahuje definici (nikoliv referenci) těchto jmen.

Výhodou této reprezentace je navíc snadné provádění redukcí respektive aplikací procesů, kdy stačí pouze změnit názvy jmen v tabulce odpovídajícího uzlu vstupního prefixu respektive uzlu abstrakce procesu, přičemž reference do tabulky používané ve všech ostatních uzlech zůstávají zachovány. Úskalím tohoto přístupu je případ, kdy provedením redukce dojde k úniku jména z původního rozsahu (tzv. scope extrusion). Jedná se o situaci, kdy je jméno definované uzlem restikce přemístěno ze své původní pozice před společného předchůdce uzlu vstupního i výstupního prefixu. Taková modifikace má však destruktivní dopad na použitý systém tabulek jmen a referencí.

Reprezentace pomocí globální tabulky jmen

Finální reprezentace jmen vychází z myšlenky definičních tabulek s tím rozdílem, že existuje vždy pouze jediná globální tabulka jmen společná pro všechny výrazy i definice procesů specifikované uživatelem. Globální tabulka jmen obsahuje záznam pro každé definované jméno v podobě jeho textového označení a dalších pomocných příznaků jako například, zda se jedná o privátní jméno vytvořené operací restrikce, či zda jeho označení bylo nahrazeno unikátním označením v důsledku prováděných operací. Všechny uzly stromu výrazu pak obsahují pouze referenci jména odpovídající indexu do této globální tabulky. Je nutné pamatovat na to, že při vytváření kopie výrazu (např. při replikaci či aplikaci definice procesu) je nutné vytvořit v tabulce nový záznam pro všechna jména uvnitř daného výrazu definovaná v rámci operace restrikce či vstupního prefixu.

Na obrázku 5.8 je znázorněna interní reprezentace výrazu $\bar{x}(\nu x)\bar{y}\langle x\rangle.0 \mid y(n).\bar{n}.0$ s použitím globální tabulky jmen.



Obrázek 5.8: Interní reprezentace výrazu $\bar{x}(\nu x)\bar{y}\langle x\rangle.0 \mid y(n).\bar{n}.0$ s využitím globální tabulky jmen.

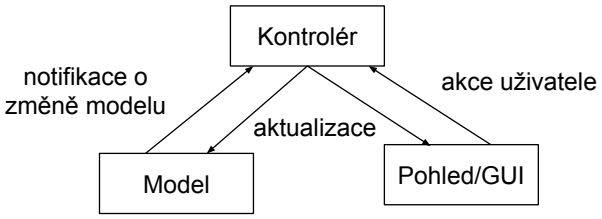
Značnou výhodou této reprezentace je její jednoduchost (v porovnání se systémem tabulek) a vyřešení problému s únikem jména z rozsahu při redukci. Dvě porovnávaná jména

jsou totožná právě tehdy, když jsou shodné jejich reference, neboli mají stejný záznam v globální tabulce jmen.

Naopak nevýhodou oproti předchozí reprezentaci je fakt, že při provedení redukce či konretizaci procesu, je nutné projít celý podstrom výrazu a v každém uzlu přemapovat příslušné reference. Dále je také vhodné zmínit, že globální tabulka může nabývat větších rozměrů oproti lokálním tabulkám a vyhledávání v ní tedy může být časově náročnější. Jedná se však o přijatelnou cenu za jednodušší strukturu a udržovatelnost.

5.5 Architektura aplikace

Architektura vytvořené aplikace je postavena na architektonickém vzoru MVP (Model-View-Presenter), derivátu MVC (Model-View-Controller). Kontrolér v tomto případě představuje řídící jednotku, která reaguje na akce uživatele a dle přijatých požadavků upravuje model. Z modelu pak kontrolér dostává notifikace o změně, na které reaguje aktualizací pohledu (uživatelského rozhraní). Schéma propojení základních komponent je znázorněno na obrázku 5.9.



Obrázek 5.9: Propojení základních prvků aplikace.

Kromě základní trojice model-pohled-kontrolér je však aplikace tvořena ještě dalšími komponentami a službami. Obrázek 5.10 znázorňuje detailnější pohled na propojení jednotlivých funkčních prvků uvnitř aplikace.

5.5.1 Kontrolér

Kontrolér je hlavním řídícím prvkem aplikace, jenž slouží jako prostředník mezi uživatelem, interními daty a viditelnou částí aplikace. Kontrolér reaguje na interakci uživatele s grafickým uživatelským rozhraním změnou stavu modelu a pohledu. Jako pomocnou službu využívá analyzátor textových příkazů a správce souborů.

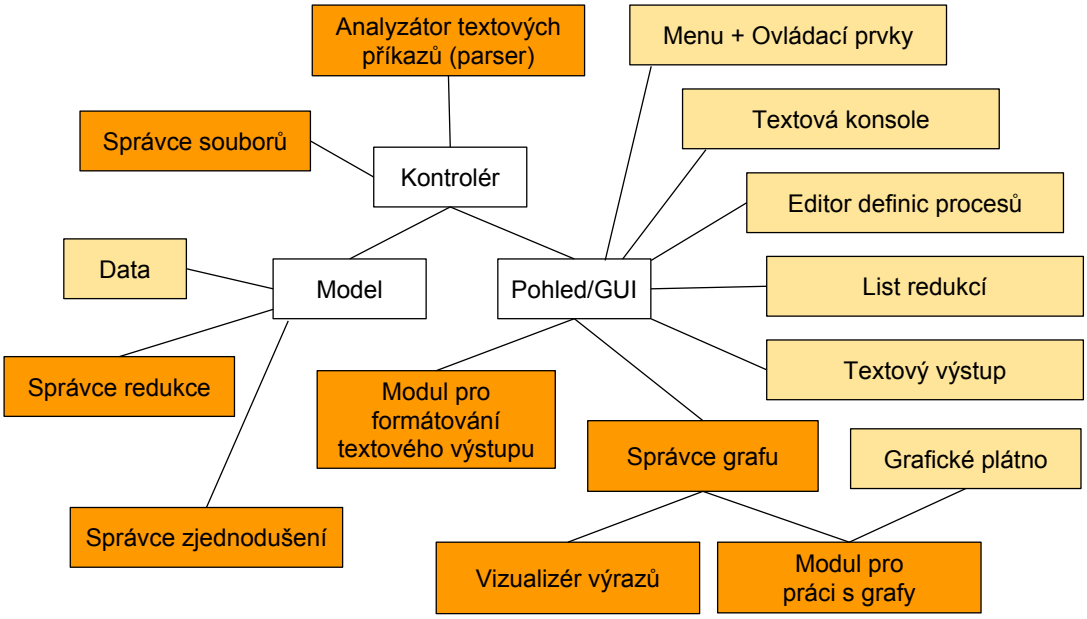
Analyzátor textových příkazů provádí lexikální a syntaktickou analýzu příkazů zadaných uživatelem do textové konsole nebo editoru definovaných procesů a zajišťuje také převod výrazů π-kalkulu z textové do interní reprezentace.

Správce souborů slouží pro ukládání a načítání aplikačních dat do nebo ze souboru.

5.5.2 Model

Model aplikace je tvořen datovou a logickou částí. Datová část uchovává aplikační data definující aktuální stav aplikace. Tato data mohou být uložena do souboru a opětovně načtena za účelem obnovení určitého stavu aplikace.

Model pracuje výhradně s interní reprezentací výrazů π-kalkulu. Součástí dat je



Obrázek 5.10: Vizualizace vztahů mezi jednotlivými moduly aplikace.

- seznam definovaných procesů,
- aktuálně vizualizovaný výraz π-kalkulu,
- globální tabulka jmen a
- uzly prefixů vybrané k redukci (redex).

Logická část poté poskytuje funkce pro modifikaci a práci s datovou částí modelu. Model používá za účelem rozšíření funčnosti správce redukcí a správce zjednodušení.

Správce redukcí poskytuje funkce pro vyhledání potenciálních redukcí a pro provedení redukce.

Správce zjednodušení pak poskytuje funkce pro zjednodušení výrazu, které spočívá v odstranění výpočetně nerelevantních částí výrazu.

5.5.3 Pohled (GUI)

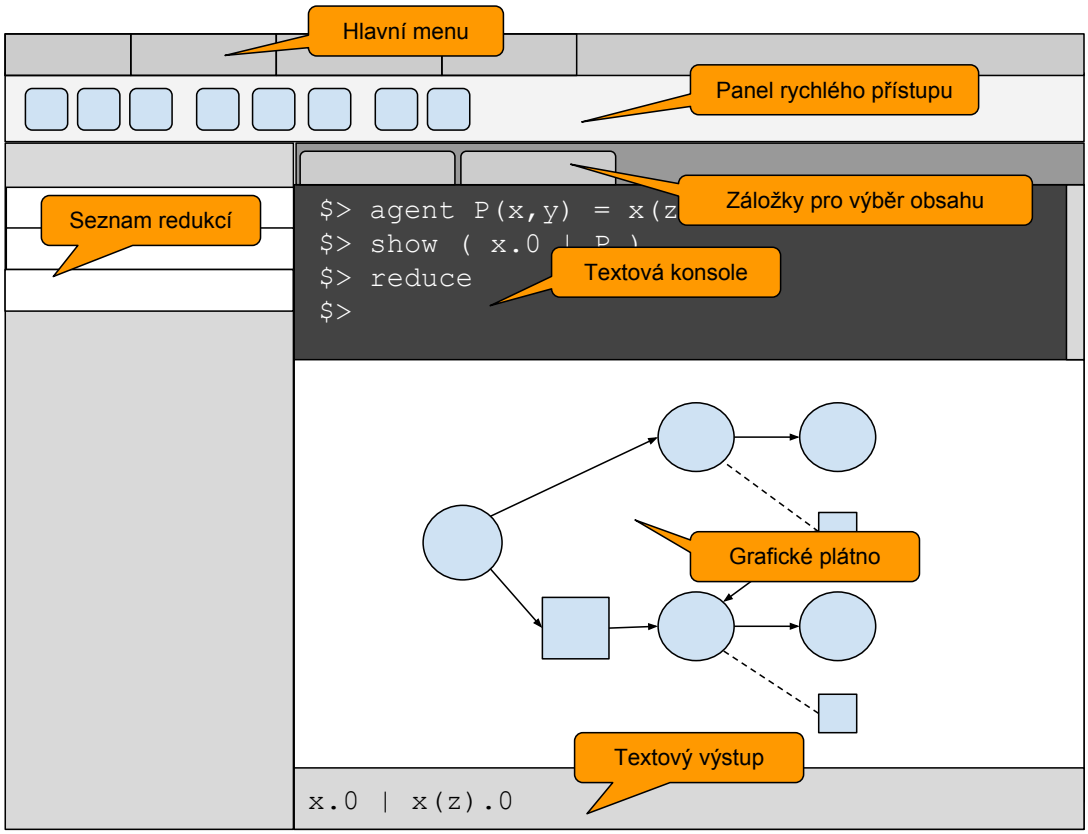
Pohled obsahuje jak pasivní komponenty, které tvoří součást grafického uživatelského rozhraní, tak i pomocné aktivní moduly, které zajišťují převod dat z interní reprezentace na grafickou či textovou reprezentaci.

Grafické uživatelské rozhraní aplikace je tvořeno následujícími komponentami:

- menu a další ovládací prvky (panel rychlého přístupu, tlačítka, apod.) – umožňují uživateli ovládat aplikaci prostřednictvím myši
- textová konsole – umožňuje uživateli ovládat aplikaci prostřednictvím textového vstupu
- editor definic procesů – umožňuje uživateli prohlížet a upravovat definované procesy

- seznam redukcí – zobrazuje proveditelné redukce nad vizualizovaným výrazem a umožňuje uživateli vybrat příslušnou redukci
- textový výstup – zobrazuje textovou reprezentaci aktuálně vizualizovaného výrazu
- grafické plátno – zobrazuje grafickou reprezentaci výrazu

Obrázek 5.11 znázorňuje navržené rozmístění výše uvedených komponent. Záložky pro výběr obsahu umožňují přepínat mezi zobrazením textové konsole a editoru definic procesů.



Obrázek 5.11: Návrh uspořádání komponent uvnitř grafického uživatelského rozhraní.

Obsah grafického plátna je definován správcem grafu. **Správce grafu** zajišťuje převod interní reprezentace výrazu na grafickou. K danému účelu využívá služby vizualizéru, který zpracovává interní reprezentaci výrazu a informuje správce grafu o potřebě vytvoření grafického uzlu či hrany. Grafické objekty jsou pak vykresleny na grafické plátno pomocí použité grafické knihovny.

Obdobně jako správce grafu zajišťuje převod z interní reprezentace na grafickou, **Modul pro formátování textového výstupu** zajišťuje převod interní reprezentace výrazu na textovou reprezentaci, rozšířenou navíc o formátovací příznaky. Formátovací příznaky definují sémantiku jednotlivých částí výstupního textového řetězce, čímž umožňují vizualizovat tyto části textu s použitím určitého stylu (např. barvy) dle jejich významu.

Kapitola 6

Implementace

Ještě před začátkem vývoje navržené aplikace bylo potřeba učinit rozhodnutí týkající se použité aplikační platformy a implementačního jazyka. Při této volbě byly zohledněny následující požadavky na výslednou aplikaci vyplývající z návrhu:

- způsobilost k provádění výpočetně náročných operací (např. vyhledávání redukcí)
- existence grafického uživatelského rozhraní
- dostupnost širokému okruhu uživatelů (pokud možno bez omezení na použitý OS nebo specifický software)

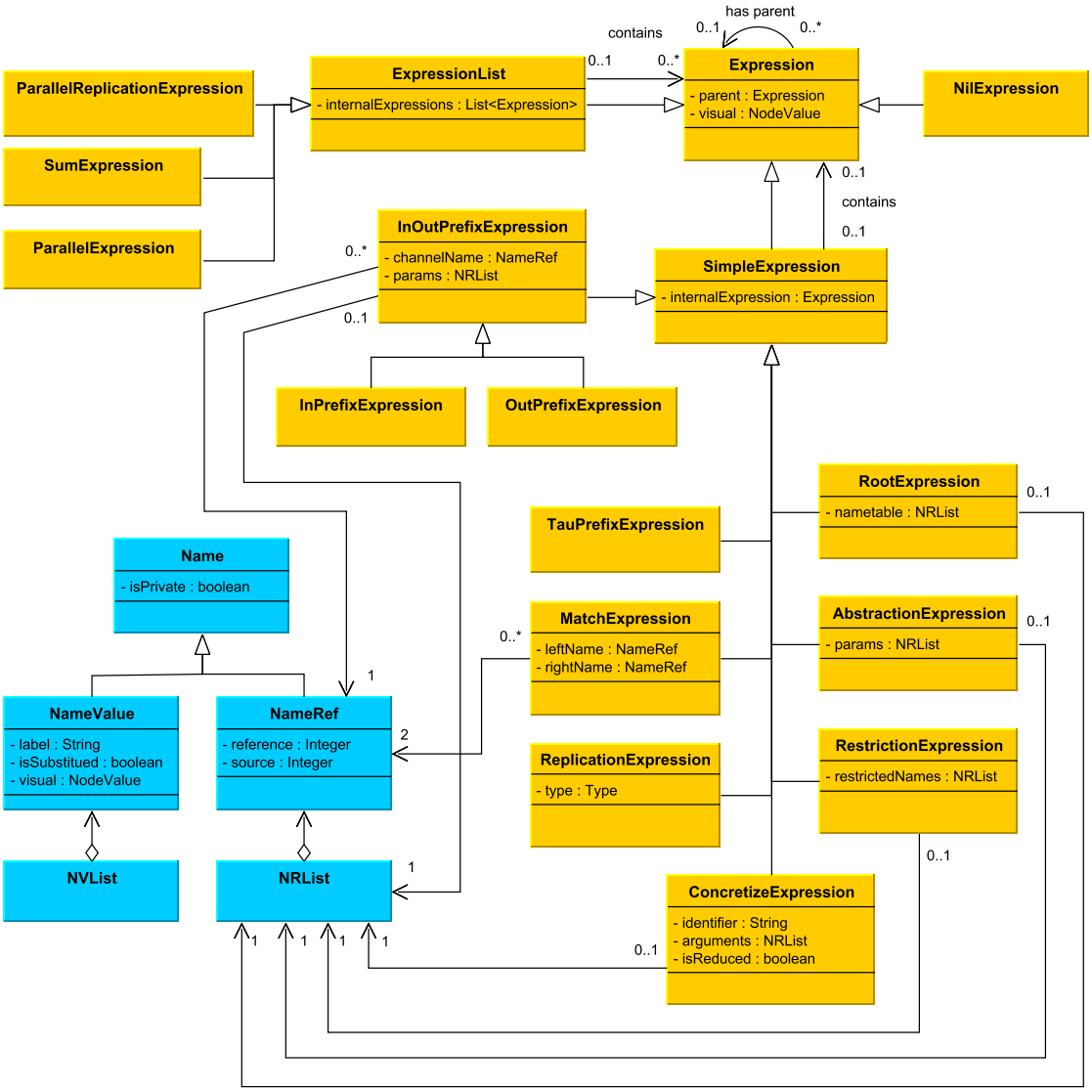
S ohledem na uvedené požadavky byla nakonec zvolena varianta desktopové aplikace implementované v jazyce Java.

V předchozí kapitole byly okrajově popsány komponenty zajišťující klíčové funkce aplikace. Tyto komponenty budou nyní detailněji analyzovány z implementačního hlediska, přičemž budou zmíněny použité nástroje a algoritmy.

6.1 Interní reprezentace

Navržená interní reprezentace výrazů byla implementována s využitím konceptu třídní dědičnosti. Nejvýše postavenou třídou v třídní hierarchii je abstraktní třída `Expression` reprezentující obecný uzel stromu a třída `Name`, reprezentující jméno. Každý typ uzlu je poté svázán s odpovídající třídou (dle tabulky 6.1), která je specializací třídy `Expression`. Obdobně třída `Name` je generalizací třídy `NameValue`, která představuje záznam o jméně uložený v globální tabulce jmen, a třídy `NameRef`, jež představuje referenci na dané jméno, tj. index do tabulky jmen. Zjednodušený diagram tříd (bez metod) znázorňující vztahy mezi třídami uzlů a jmen je uveden na obrázku 6.1.

Za účelem možného rozšíření funkcionality tříd uzlů a snadného procházení stromu výrazu byla vytvořena abstraktní třída `ExpressionVisitor`. Tato třída je postavena na myšlence návrhového vzoru Návštěvník – tj. implementuje metodu `visit`, která dle typu přijaté instance provede odpovídající akce. Tato třída má v implementovaném řešení uplatnění jako nadřazená třída mnoha dalších tříd, jejichž účlem je například převod výrazů z interní reprezentace na textovou či grafickou reprezentaci.



Obrázek 6.1: Redukovaný diagram tříd interní reprezentace výrazu. Třídy reprezentující uzly stromu jsou znázorněny oranžovou barvou, třídy reprezentující jména nebo seznamy jmen jsou vykresleny modře.

Typ uzlu	Třída
Nil (prázdná suma)	`NilExpression`
Nepozorovatelný prefix	`TauPrefixExpression`
Vstupní prefix	`InPrefixExpression`
Výstupní prefix	`OutPrefixExpression`
Prefix shody	`MatchExpression`
Restrikce	`RestrictionExpression`
Suma	`SumExpression`
Paralelní kompozice	`ParallelExpression`
Paralel. kompozice (repl.)	`ParallelReplicationExpression`
Replikace	`ReplicationExpression`
Konkretizace procesu	`ConcretizeExpression`
Abstrakce procesu	`AbstractionExpression`
Kořen výrazu	`RootExpression`

Tabulka 6.1: Mapování uzlů interní reprezentace výrazů π-kalkulu na třídy.

6.2 Převod výrazu z textové do interní reprezentace

Jedním z implementačních problémů navržené aplikace bylo zpracování textového vstupu (z konsole nebo editoru definic procesů). Prostřednictvím textového vstpu může uživatel specifikovat výrazy určené k vizualizaci, definovat procesy, ale také aplikaci ovládat.

Tabulka 6.2 obsahuje přehled všech příkazů, které může uživatel zadat prostřednictvím konzole a provést tak odpovídající akci.

Příkaz	Odpovídající akce
agent ⟨*definice procesu*⟩	přidá definovaný proces do seznamu procesů
show ⟨*výraz*⟩	vizualizuje specifikovaný výraz
env ⟨*identifikátor procesu*⟩	vypíše definici specifikovaného procesu
env	vypíše všechny definice procesů
redlist	zobrazí list možných redukcí pro vizualizovaný výraz
reduce	provede redukci vybraných uzlů vizualizovaného výrazu
simplify	provede zjednodušení vizualizovaného výrazu
clear	vyčistí obsah textové konsole
reset	uvede aplikaci do výchozího stavu
help	vypíše textovou nápovědu
quit	ukončí aplikaci
exit	ukončí aplikaci

Tabulka 6.2: Přehled textových příkazů použitelných pro ovládání aplikace.

Pro účely analýzy textového vstupu byl použit nástroj **ANTLR**[1] (verze 4.5.3). Jedná se o generátor překladačů, který na základě specifikované gramatiky vygeneruje třídy poskytující metody pro lexikální a syntaktickou analýzu vstupního textu. Výstupem této analýzy je derivační strom, který může být dále zpracováván. Kromě samotného analyzátoru je při vhodné konfiguraci vygenerováno také rozhranní `Visitor`. Třída, která implementuje toto

[1] http://www.antlr.org/

rozhraní pak může procházet derivační strom a nad každým navštíveným uzlem provést odpovídající akce – například vytvořit objekt některé z tříd interní reprezentace výrazu. Využitím tohoto principu v aplikaci je uživatelem zadaný výraz převeden z textové do interní reprezentace. Gramatika použitá pro generování analyzátoru je uvedena v příloze A.

6.3 Převod výrazu z interní do textové reprezentace

Aplikace byla navržena tak, aby kromě grafického výstupu poskytovala i výstupy textové prezentující jak textovou reprezentaci aktuálně zpracovávaného výrazu, tak i definice procesů (v editoru procesů). Za tímto účelem bylo nutné vytvořit aplikační modul, který by realizoval převod výrazů z interní reprezentace do textové. Implementací takového modulu je třída `TextFormater`, která dědí ze třídy `ExpressionVisitor`, popsané dříve.

Tato třída obsahuje metodu `getString`, která inicializuje rekurzivní průchod stromem interní reprezentace výrazu. Během tohoto průchodu jsou při návštěvě určitých uzlů vytvářeny odpovídající objekty třídy `TextBlob`, které jsou ve správném pořadí přidávány do výstupního seznamu. Hlavním atributem třídy `TextBlob` je textový řetězec, který odpovídá určitému termu jazyka π-kalkulu. Dodatečnými atributy jsou pak formátovací příznaky, které specifikují význam daného termu – například zda se jedná o jméno nebo zda je term součástí vybraného uzlu. Tyto příznaky poté umožňují příslušným komponentám grafického uživatelského rozhranní vizualizovat jednotlivé bloky textu specifickým stylem (např. modrou barvou).

Obrázek 6.2 demonstruje výše popsaný princip pro výraz, jenž je součástí definice procesu P. V rámci zjednodušení jsou v objektech třídy `TextBlob` znázorněny pouze dva formátovací příznaky, a to příznak jména a příznak procesu.

Při průchodu stromem a vytváření textových bloků je nutné brát ohled na pravidla precedence operátorů, neboť součástí interní reprezentace není uzel, který by explicitně vynucoval přítomnost závorek. Vytváření textových bloků pro závorky se řídí několika pravidly.

Zpracovávaný uzel/výraz bude uzávorkován v následujících případech:

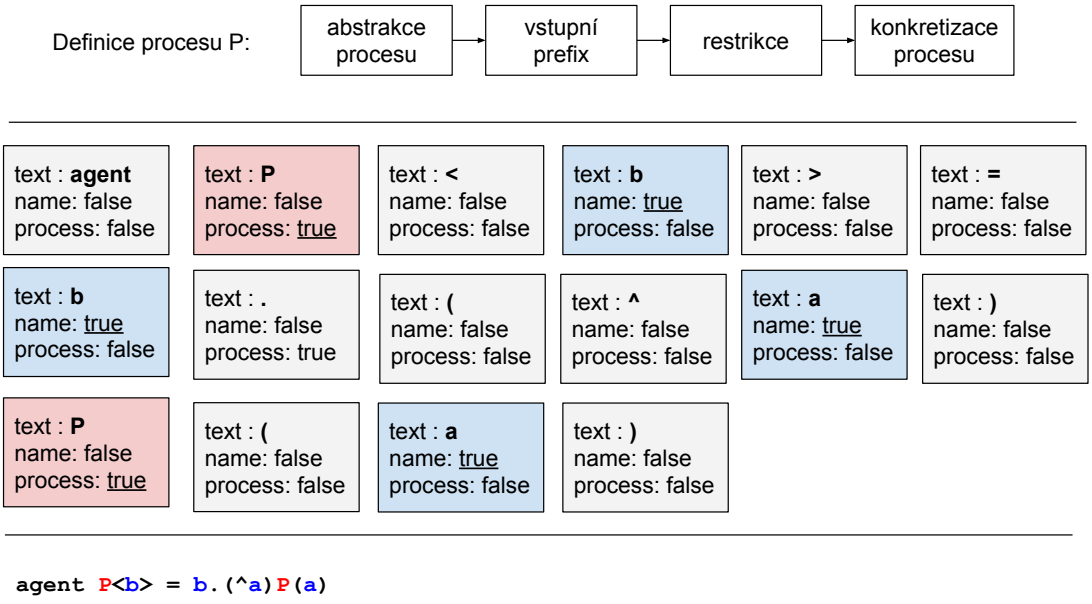
- je-li objektem třídy `RestrictionExpression`
- je-li objektem třídy `SumExpression` a některý z jeho předchůdců má viditelnou textovou reprezentaci
- je-li objektem třídy `ParallelExpression` (nebo `ParallelReplicationExpression`, který má být vizualizován) a jeho nejbližším předchůdcem, který má viditelnou textovou reprezentaci, není objekt třídy `SumnExpression`

6.4 Převod výrazu z interní do grafické reprezentace

Převod interní reprezentace výrazu na grafickou, je v aplikaci zajišťován prostřednictvím správce grafu, který pro daný účel využívá takzvaný vizualizér a modul pro práci s grafy.

6.4.1 Modul pro práci s grafy

Vykreslování grafických entit (uzlů a hran) na plátno grafu (grafický výstup) je realizováno s využitím externí grafické knihovny určené pro tvorbu grafů. Za účelem zajištění nezá-



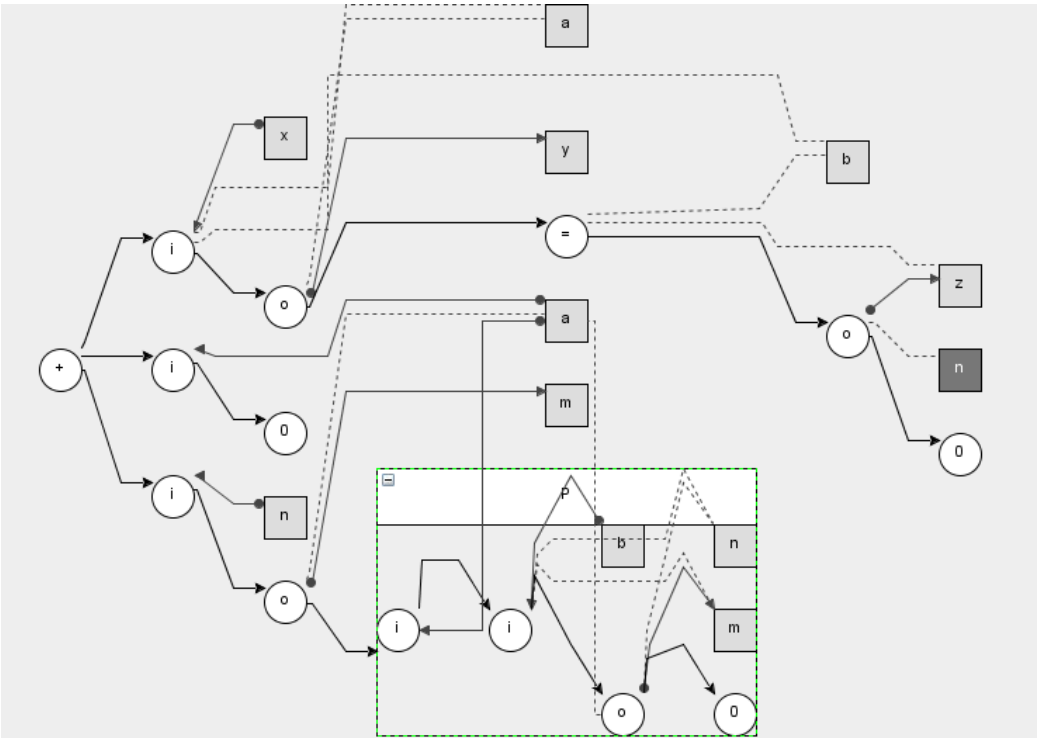
Obrázek 6.2: Princip převodu interní reprezentace výrazu na textovou s použitím seznamu textových bloků. Nahoře je zobrazena interní reprezentace výrazu jako součást definice procesu P, uprostřed odpovídající seznam objektů třídy `TextBlob`, dole naformátovaná textová reprezentace výrazu.

vislosti aplikace na konkrétní implementaci knihovny, bylo vytvořeno rozhraní `GraphLib`, které deklaruje metody sloužící pro modifikaci obsahu plátna grafu nebo získávání informací o grafických entitách. Toto rozhraní je implementováno obalovací třídou příslušné knihovny, čímž je vytvořena jednotná abstraktní vrstva nad rozdílnými aplikačními rozhraními různých knihoven. Pro možnost předávání informací o akcích uživatele prováděných nad jednotlivými grafickými entitami nebo plátnem grafu pak bylo vytvořeno rozhraní `GraphListener`. Toto rozhraní je implementováno správcem grafu, který je tak prostřednictvím metod daného rozhraní informován o interakcích uživatele s grafickou reprezentací.

Jednou z výhod výše popsaného principu založeného na implementaci rozhraní je nahraditelnost použité knihovny pro práci s grafy libovolnou knihovnou s obdobnou funkcionalitou bez výraznějších zásahů do zdrojového kódu aplikace. Prostřednictvím rozhranní jsou navíc předávány pouze informace o typu grafických entit, nikoliv přesná specifikace vzhledu. To umožňuje přizpůsobit grafickou reprezentaci výrazu možnostem konkrétní knihovny.

Během vývoje aplikace byly pro srovnání použity dvě různé knihovny třetích stran – open source knihovna **JGraphX**[2] (verze 3.7.2) a komerční knihovna **yFiles**[3] (verze 3.0.0). Pro každou z těchto knihoven byla vytvořena obalující třída (wrapper) implementující rozhranní `GraphLib`. Ukázalo se, že knihovna JGraphX má v použité verzi 3.7.2 značné problémy s hierarchickým pozicováním uzlů grafu. Ty se nakonec podařilo částečně vyřešit použitím zdrojových kódů z verze 1.3.1.6 a speciálním přístupem, kdy je každá vrstva hierarchie pozicována zvlášť. Při hierarchickém uspořádání však i tak v některých případech dochází k překryvu uzlů hranami a může být tedy nutná manuální úprava ze strany uživatele spočíva-

[2] https://github.com/jgraph/jgraphx
[3] https://www.yworks.com/products/yfiles-for-java



Obrázek 6.3: Nejlepší dosažený výsledek automatického pozicování uzlů s použitím aktuální verze knihovny JGraphX.

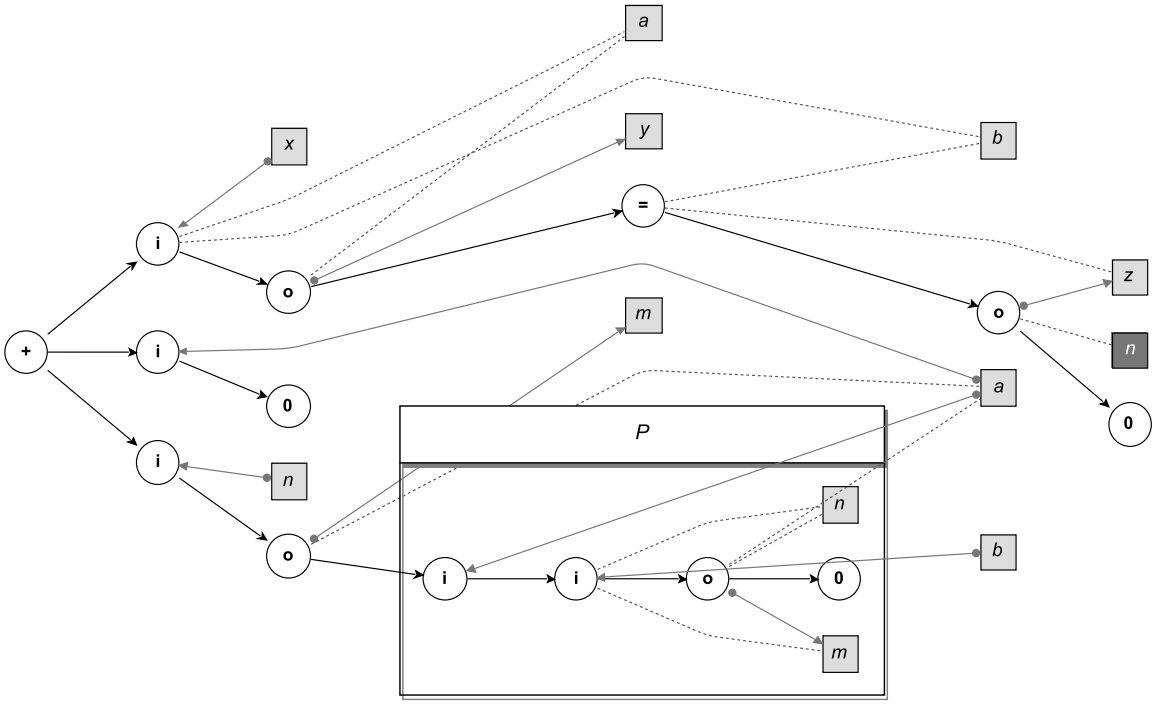
jící v přesunutí některých uzlů či hran. Knihovna yFiles naproti tomu vykazuje velmi kvalitní výsledky, co se týče automatického pozicování uzlů. Porovnání výstupů knihovny JGraphX a yFiles pro komplexnější výraz $x(a,b).\bar{y}\langle a\rangle.(\nu n)[b=z]\bar{z}\langle n\rangle.0 + a.0 + n.\bar{m}\langle a\rangle.P\langle a,b\rangle$, kde $P(q,r) = q.r(n,m).\bar{m}\langle n,q\rangle.0$, je znázorněno na obrázcích 6.3, 6.4 a 6.5.

Při vývoji aplikace byla využita 60-ti denní evaluační licence knihovny yFiles. Z toho důvodu je součástí odevzdané práce pouze modul implementující rozhranní `GraphLib` určený pro práci s knihovnou yFiles, nikoliv však knihovna samotná.

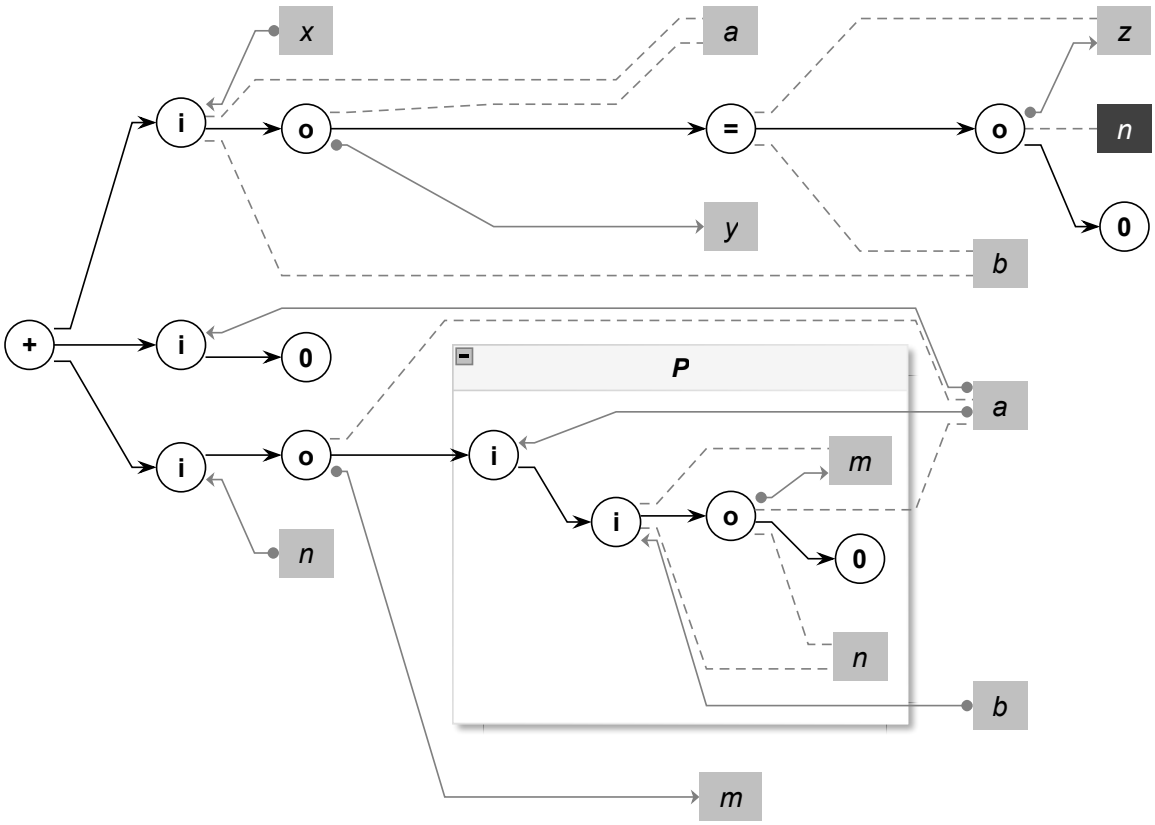
6.4.2 Vizualizér

Vizualizér implementovaný třídou `Visualizer`, která dědí ze třídy `ExpressionVisitor`, poskytuje metody pro průchod stromem interní reprezentace. Při návštěvě jednotlivých uzlů vizualizér generuje objekty třídy `NodeValue` respektive `EdgeValue`, které obsahují pomocné informace pro vizualizaci uzlu respektive hrany (např. textové označení, typ, příznak viditelnosti, příznak výběru apod.). Při vytváření těchto objektů pro uzly konkretizace procesu bere vizualiér ohled jednak na zvolený styl grafické reprezentace (lineární nebo hierarchická), tak i na to, zda je daný proces definován. Invokací metod z rozhraní /VisualListener pak vizualizér žádá správce grafu o vytvoření odpovídajících grafických objektů (uzlů nebo hran). Jako parametr těchto metod předává nejen vizuální informace pro daný objekt, ale také topologický kontext – tj. objekt třídy `NodeValue` rodičovského objektu a v případě hrany i zdrojového a cílového objektu.

Pokud navštívený uzel stromu interní reprezentace již obsahuje uložený objekt třídy `NodeValue`, nový objekt se nevytváří a namísto něj se použije stávající. Taková situace může



Obrázek 6.4: Výsledek automatického pozicování uzlů s použitím starší verze knihovny JGraphX.



Obrázek 6.5: Výsledek automatického pozicování uzlů s použitím knihovny yFiles.

nastat například při překreslení grafické reprezentace nebo vizualizaci výrazu načteného ze souboru.

6.4.3 Správce grafu

Jak již bylo popsáno výše, vizualizér slouží k analýze interní reprezentace výrazu a vyhodnocení jaké uzly a hrany mají být vytvořeny. Modul pro práci s grafem je pak na základě vytvořených vizuálních informací schopen vykreslit odpovídající grafické entity na plátno grafu. Správce grafu implementovaný třídou `GraphManager` přitom zajišťuje správné propojení obou procesů a předávání vizuálních informací.

Správce grafu se aktivně podílí na utváření grafické reprezentace výrazů a na jejích úpravách. Kromě řízení převodu výrazu z interní do grafické reprezentace, ať už při vizualizaci výrazu jako celku nebo jen jeho části (např. replikované větve), zpracovává také požadavky na modifikaci vytvořené grafické reprezentace (např. požadavek na zvýraznění některých uzlů). Upravuje přitom objekty třídy `NodeValue` a `EdgeValue` nastavením relevantních příznaků tak, aby při případném překreslení grafické reprezentace zůstala zachována informace o stavu jednotlvých uzlů a bylo tak možné zrekonstruovat její původní podobu. Správce grafu rovněž rozhoduje o tom, kdy je nutné provést automatické rozmístění uzlů tak, aby bylo provedeno vždy pouze jednou, a to jen když je potřeba. Implementací rozhraní `GraphListener` je pak správce grafu rovněž schopen přijímat informace o interakcích uživatele s grafickými objekty. Příkladem takové interakce může být kliknutí levého či pravého tlačítka myši na uzel, použití tlačítka pro rozbalení a sbalení skupiny uzlů a podobně. Na základě těchto informací je schopen na danou akci uživatele adekvátně reagovat nebo o ní informovat kontrolér.

Správce grafu představuje ve své podstatě prostředníka mezi libovolnou použitou knihovnou, pracující s grafickými objekty neurčité třídy, a ostatními moduly jádra aplikace, pracujícími s objekty navržené interní reprezentace výrazů. Za účelem podpory transformací grafických entit na uzly a jména stromu interní reprezentace výrazu (a naopak), bylo nutné zavést jednoznačné mapování těchto objektů. To se nakonec podařilo zajistit s využitím objektů třídy `NodeValue`, které byly rozšířeny o jedinečný číselný identifikátor. Tyto objekty jsou při jejich vzniku ukládány jako součást odpovídajícího objektu interní reprezentace i jako součást grafického objektu, čímž je zajištěna jednoznačná identifikovatelnost těchto objektů. V případě, že zvolená knihovna pro práci s grafy nepodporuje ukládání uživatelského objektu uvnitř grafických objektů, je nutné, aby tuto funkcionalitu implementovala příslušná obalovací třída knihovny. Identifikátor třídy `NodeValue` pak slouží mimo jiné i jako index do mapovací tabulky, která je součástí správce grafu. V ní je na konkrétním indexu uložen vždy odpovídající objekt interní reprezentace společně s odpovídajícím grafickým objektem. Správce grafu, je tak kdykoliv schopen určit, kterému objektu interní reprezentace odpovídá uživatelem zvolený grafický objekt (např. při kliknutí) nebo naopak, který grafický objekt má být vizuálně upraven, pokud o to žádá kontrolér (např. jako důsledek výběru prefixu k redukci).

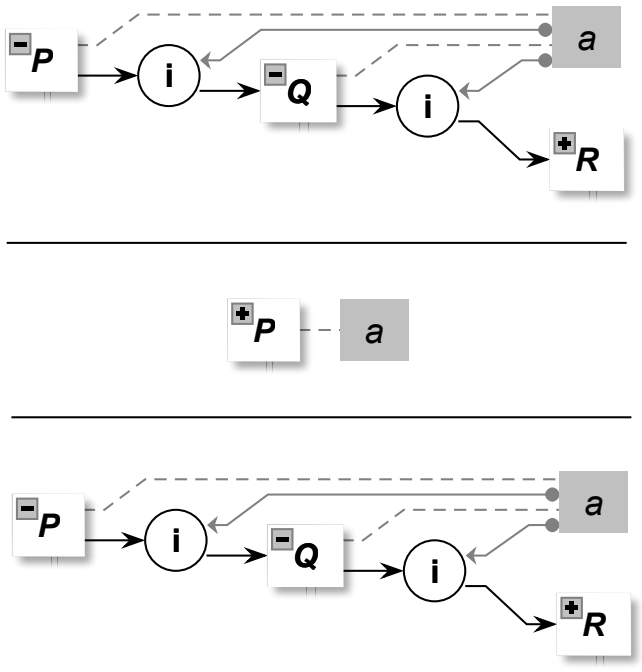
Významnou roli hraje správce grafu v případě, že součástí vizualizovaného výrazu jsou aplikace definovaných procesů. Aplikace definovaného procesu je reprezentována speciálním uzlem grafu, jehož součástí je malé tlačítko určené k expanzi či sbalení daného uzlu. Ve výchozím nastavení se každý uzel procesu nachází ve sbaleném stavu a tlačítko je označeno symbolem +. Kliknutím na toto tlačítko dává uživatel pokyn k vizualizaci výrazu daného procesu. Podle zvoleného stylu grafické reprezentace je výraz procesu vizualizován formou podgrafu vloženého buď přímo do vybraného uzlu procesu (při hierarchickém stylu), nebo

připojeného za něj (při lineárním stylu). Správce grafu reaguje na požadavek expanze (v případě hierarchického stylu grafické reprezentace) následovně:

1. Ověří, zda byl již výraz daného procesu dříve vizualizován.
2. Pokud ne, požádá kontrolér o přidělení výrazu instance procesu. Přidělený výraz následně vizualizuje invokací příslušné metody vizualizéru a zajistí správné připojení vytořeného podgrafu k již existující grafické reprezentaci.
3. Pokud ano, nastaví všem potomkům uzlu procesu příznak viditelnosti (rekurzivně i jejich potomkům) a invokuje metodu pro zviditelnění uzlu z rozhranní modulu pro práci s grafy.
4. Nastavením příznaku viditelnosti a invokací příslušné metody skryje hrany připojující k uzlu procesu uzly parametrů. (Parametry procesu jsou u hierarchického stylu grafické reprezentace navázány přímo k interním uzlům procesu.)

Mírně odlišný postup je použit v případě lineárního stylu grafické reprezentace, kdy ve třetím bodě není příznak viditelnosti nastavován potomkům uzlu, ale rekurzivně všem následníkům uzlu procesu. Čtvrtý bod se pak u lineárního stylu neuplatní vůbec.

Obráceným postupem zpracovává správce grafu požadavky na sbalení uzlů. Při skrývání částí grafu bere ohled na to, aby v grafické reprezentaci nezůstal zobrazen žádný izolovaný uzel jména. Při zviditelňování částí grafu respektuje uložený stav jednotlivých uzlů, tedy nacházel-li se například nějaký uzel procesu v expandovaném stavu, bude po sbalení a následné expanzi jeho přímého předka opět v expandovaném stavu, jak demonstruje obrázek 6.6.



Obrázek 6.6: Nahoře je zobrazena grafická reprezentace výrazu $P\langle a\rangle$, kde $P(a) = a.Q\langle a\rangle$, $Q(a) = a.R$ a $R = 0$. Uprostřed je zobrazen stejný výraz po sbalení procesu P. Dole tentýž výraz po expanzi procesu P. Sbalení nebo expanze uzlu procesu má vliv pouze na viditelnost ostatních uzlů, nikoliv na jejich stav.

Výše popsaný princip vizualizace výrazu na vyžádání uživatelem řeší problém nekonečného rozvoje výrazu, který může nastat například když některý z definovaných procesů odkazuje sám sebe.

Přestože obě použité knihovny pro práci s grafem podporují expanzi i sbalení uzlů (při kliku na tlačítko jsou automaticky skryty či naopak zobrazeny všechny uzly, které jsou potomky daného uzlu), správce uzlu je implementován tak, že tato podpora není u knihovny vyžadována. V případě, že nějaká knihovna nepodporuje expanzi či sbalení uzlů, stačí pokud bude podporovat skrývání a zobrazování jednotlivých grafických objektů na vyžádání. Rovněž interakci s tlačítkem uzlu je možné, v případě potřeby, nahradit jinou akcí uživatele (například dvojklikem).

6.5 Práce se soubory

Dobrým zvykem většiny aplikací bývá možnost uložit vytvořenou práci do souboru za účelem pozdějšího načtení a znovuobnovení uloženého stavu aplikace. V případě grafických editorů pak zpravidla nechybí ani podpora pro export grafiky do souboru různých formátů. Vytvořené grafické dílo tak může být volně sdíleno a publikovánko, aniž by k tomu bylo potřeba danou aplikaci či editor spouštět. Obě tyto funkce jsou podoprovány i vytvořenou aplikací.

Pro uložení celkového stavu aplikace byl využit princip serializace, který umožňuje převést libovolný objekt třídy implementující rozhraní `Serializable` a obsahující atribut `serialVersionUID` na sekvenci bytů. Tato sekvence pak může být uložena do souboru a při načtení ze souboru transformována na původní objekt.

Ve vytvořené aplikaci je serializována datová část modelu, která obsahuje pouze položky nezbytné pro kompletní rekonstrukci uloženého stavu aplikace. Jedná se konkrétně o seznam definovaných procesů, vizualizovaný výraz včetně pomocných vizuálních informací uložených v objektu třídy `NodeValue`, dále globální tabulku jmen rovněž včetně vizuálních informací a nakonec redex obsahující uzly prefixů vybrané k redukci.

Nevýhodou serializace je to, že vytvořený soubor není určen pro editaci mimo vytvořenou aplikaci. Pokud by tak například uživatel chtěl pouze upravit některý z definovaných procesů, není reálné, aby změnu provedl jinak než přes aplikační rozhraní.

Aplikace však kromě serializace podporuje i export definic procesů do textového souboru. Tento soubor je pak editovatelný v libovolném textovém editoru a může být kdykoliv použit pro načtení definic procesů zpět do aplikace. Cenou za editovatelnost mimo aplikaci je však to, že do souboru jsou ukládány výhradně definice procesů a není tedy možná rekonstrukce kompletního stavu aplikace.

Seznam procesů je převáděn do textové reprezentace obdobným způsobem, jaký byl popsán v kapitole 6.3 s tím rozdílem, že není vytvářen seznam speciálních objektů s formátovacími příznaky, ale pouze jednoduchý textový řetězec. Při načítání souboru se pak uplatní analyzátor textového vstupu, popsaný v kapitole 6.2.

Poslední variantou generování souborového výstupů z aplikace je export grafické reprezentace. Vytvořená aplikace podporuje v současné verzi několik rastrových a několik vektorových formátů: JPEG, PNG, BMP, EMF, EPS a SVG. Množina podporovaných formátů a implementace kódování částečně závisí na použité knihovně pro práci s grafy. Některé z nich totiž nabízí pomocné metody pro export grafu přímo jako součást svého aplikačního rozhraní. V případě knihovny JGraphX se jedná například o statickou metodu `createBufferedImage` třídy `mxCellRenderer`, která realizuje kódování obsahu plátna grafu do některého z podporovaných bitmapových formátů. Použitím metody `write` třídy

`ImageIO` pak může být vytvořená bitmapa zapsána do souboru. Třída `mxCellRenderer` poskytuje rovněž metodu `createSvgDocument` pro kódování do formátu SVG. Obdobou třídy `mxCellRenderer` je v případě knihovny yFiles třída `PixelImageExporter` realizující export grafu do bitmapových formátů. Na rozdíl od třídy `mxCellRenderer` však nepodporuje vytváření SVG souborů. Pro tyto účely je v aplikaci použita externí knihovna **Batik**[4] poskytující metody pro práci s SVG dokumenty. Obdobně pro export do formátu EMF a EPS pak byla v obou případech použita externí knihovna **FreeHEP**[5].

6.6 Algoritmus vyhledávání redukcí

Aplikace byla navržena tak, aby kromě vizualizace výrazů π-kalkulu umožňovala provádět nad vizualizovaným výrazem (respektive jeho interní reprezentací) redukce. Provedení redukce je podmíněno existencí takzvaného redexu ve výrazu, kdy každý redex je tvořen buď nepozorovatelným prefixem, nebo dvojicí vstupního a výstupního prefixu, přičemž musí být dodržena pravidla redukce specifikovaná v kapitole 3.3.

Jelikož aplikační modul pro práci s redukcemi implementovaný třídou `ReductionManager` pracuje výhradně s interní reprezentací výrazu, definujeme nyní několik pomocných pojmů, které se vztahují k interní stromové reprezentaci výrazu, a s jejichž pomocí bude následně možné popsat princip vyhledávání redukovatelných uzlů i provádění redukcí.

Definice 6.6.1. *Dvě jména jsou **shodná**, právě když jejich reference (objekty třídy `NameRef`) odkazují na stejný záznam v globální tabulce jmen.*

Definice 6.6.2. *Uzel vstupního prefixu (`InPrefixExpression`) a uzel výstupního prefixu (`OutPrefixExpression`) jsou **komplementární**, právě když jejich jména označující komunikační kanál jsou shodná a když souhlasí i počet jejich parametrů.*

Definice 6.6.3. *Uzel prefixu shody (`MatchExpression`) je **platný**, právě když jsou jeho porovnávaná jména shodná.*

Definice 6.6.4. *Uzel je **blokující**, právě když se jedná o uzel nepozorovatelného prefixu (`TauPrefixExpression`), uzel vstupního prefixu (`InPrefixExpression`), uzel výstupního prefixu (`OutPrefixExpression`), nebo neplatný uzel prefixu shody (`MatchExpression`).*

Definice 6.6.5. *Uzel je **redukovatelný** právě když platí jedno z následujících tvrzení:*

1. *Jedná se o uzel nepozorovatelného prefixu (`TauPrefixExpression`) a žádný z jeho předchůdců není blokujícím uzlem.*
2. *Jedná se o uzel vstupního prefixu (`InPrefixExpression`) I, pro který existuje ve stromu stejného výrazu komplementární uzel (`OutPrefixExpression`) O takový, že ve stromu existuje uzel paralelní kompozice (`ParallelExpression` nebo `ParallelReplicationExpression`), který je společným předchůdcem uzlu I i uzlu O a zároveň ve stromu neexistuje blokující uzel, který by byl předchůdcem uzlu I nebo uzlu O.*

[4] https://xmlgraphics.apache.org/batik/
[5] http://java.freehep.org/

3. *Jedná se o uzel výstupního prefixu (`OutPrefixExpression`) O, pro který existuje ve stromu stejného výrazu komplementární uzel (`InPrefixExpression`) I takový, že ve stromu existuje uzel paralelní kompozice (`ParallelExpression` nebo `ParallelReplicationExpression`), který je společným předchůdcem uzlu I i uzlu O a zároveň ve stromu neexistuje blokující uzel, který by byl předchůdcem uzlu I nebo uzlu O.*

Definice 6.6.6. *Redex je **kompletní**, právě když je tvořen uzlem nepozorovatelného prefixu (`TauPrefixExpression`) nebo dvojicí uzlů vstupního (`InPrefixExpression`) a výstupního (`OutPrefixExpression`) prefixu, které jsou vzájemně komplementární, žádný z jejich předchůdců není blokujícím uzlem a existuje společný předchůdce, jenž je uzlem paralelní kompozice (`ParallelExpression` nebo `ParallelReplicationExpression`).*

Pro výše uvedené definice byly v aplikaci implementovány odpovídající metody, jejichž smyslem je ověřit konkrétní vlastnost daného jména, uzlu či redexu.

Zřejmě platí, že redukci je nad interní reprezentací výrazu možné provést pouze tehdy, je-li specifikován kompletní redex. Chce-li tedy uživatel aplikace provést redukci, musí nejprve určit, které uzly mají tvořit příslušný redex. Výběr uzlů je možné provádět buď interakcí s grafem (kliknutím pravého tlačítka myši na konkrétní uzel a výběrem položky z kontextového menu) nebo výběrem položky ze seznamu možných redukcí.

Při výběru uzlu k redukci je uzel grafu obarven červeně a odpovídající uzel stromu interní reprezentace je uložen do redexu v datové části modelu aplikace. Není-li daný redex kompletní, jsou růžově obarveny všechny vizualizované komplementární uzly. Vybere-li uživatel uzel ve chvíli, kdy již redex nějaký uzel obsahuje, ověří se, zda by přidáním vybraného uzlu vznikl kompletní redex. Pokud ne, je pro vybraný uzel vytvořen nový redex jako náhrada původního.

Pro výběr redexu z listu redukcí i pro zvýraznění komplementárních uzlů při výběru z grafu je nezbytné, aby aplikace poskytovala prostředky pro generování seznamu *všech* kompletních redexů využitelných pro provedení redukce. Tato funkcionalita je implementována v metodě `generateReductionList` třídy `GraphManager`.

Vstupem algoritmu je kořenový uzel stromu interní reprezentace, jenž reprezentuje vizualizovaný výraz π-kalkulu. Výstupem je pak seznam nalezených redexů. Algoritmus pro vyhledání rekurzí je založen na myšlence rekurzivního průchodu stromem interní reprezentace výrazu. Jedná se v podstatě o upravený algoritmus prohledávání do hloubky, kdy posledním prohledávaným uzlem je buď uzel typu Nil, nebo libovolný blokující uzel.

Úskalím výše popsaného algoritmu je potenciálně nekonečný rozvoj výrazu v případě, že výraz obsahuje aplikace definovaných procesů. Modeluje-li daný výraz π-kalkulu nekonečné chování (příklad takového procesu je znázorněn na obrázku 6.7), pak může existovat nekonečně mnoho proveditelných redukcí a požadavek na vyhledání *všech* kompletních redexů tak zřejmě není realizovatelný. Proto bylo nutné tento požadavek omezit zavedením dodatečné podmínky ukončení prohledávání.

Definice 6.6.7. ***Kontextem** uzlu U je proces P, jehož uzel konkretizace je posledním uzlem konkretizace na cestě od kořenového uzlu k uzlu U. V případě, že se na cestě od kořenového uzlu k uzlu U nenachází žádny uzel konkretizace, pak je kontext uzlu prázdný.*

Definice 6.6.8. *Dvě jména jsou **zdrojově shodná**, právě když jejich zdrojové reference (objekty třídy `NameRef`) odkazují na stejný záznam v globální tabulce jmen.*

Poznámka. Zdrojová reference označuje původní jméno, z něhož analyzované jméno vzniklo. Ve většině případů je zdrojová reference jména stejná jako jeho reference. Ke změně dochází replikací jména, která je provedena u jmen definovaných operací restrikce nebo vstupního prefixu při instanciaci (aplikaci) definovaného procesu nebo při rozbalení operace replikace.

Definice 6.6.9. *Uzel U, jenž je uzlem konkretizace procesu P je **znovupoužitý**, pokud se na cestě od kořenového uzlu stromu k uzlu U vyskytuje jiný uzel V, jenž je rovněž konkretizací procesu P, se zdrojově shodnými parametry a se stejným kontextem.*

Zanořování do hloubky při prohledávání stromu interní reprezentace výrazu je ukončeno uzlem konkretizace procesu P, pokud se jedná o znovupoužitý uzel a zároveň daný uzel konkretizace nemá následníka (instance procesu P nebyla v rámci dané aplikace procesu dosud vizualizována).

Obrázek 6.7 znázorňuje výraz, v němž byly nalezeny právě tři možné redukce v důsledku vizualizace tří instancí procesu P. Vzhledem k nekonečnému rozvoji výrazu by však bez použití dodatečné podmínky zastavení algoritmus nikdy neskončil.

Účelem obrázku 6.8 je demonstrovat závislost podmínky ukončení na kontextu uzlu. V případě, že by definice znovupoužitelnosti neobsahovala podmínku na shodu kontextů, zvýrazněná redukce by nebyla algoritmem nalezena, neboť prohledávání by bylo ukončeno při návštěvě druhého uzlu konkretizace procesu P.

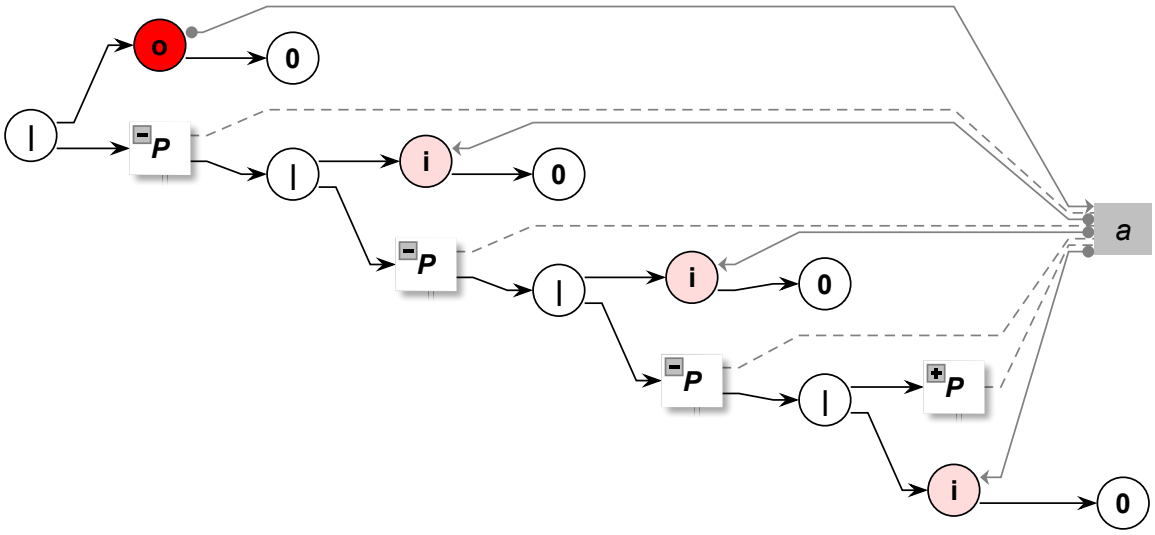
Obdobně obrázek 6.9 pak demonstruje význam závislosti podmínky ukončení na shodě parametrů. V případě, že by definice znovupoužitelnosti neobsahovala podmínku na (zdrojovou) shodu parametrů, zvýrazněné redukce by nebyly algoritmem nalezeny, neboť prohledávání by bylo ukončeno při návštěvě druhého uzlu konkretizace procesu P.

Na obrázku 6.10 je pak zobrazen výraz, pro který je nezbytné, aby parametry procesů byly porovnávány na zdrojovou shodu jmen. Jako parametr konkretizace procesu P je totiž předáváno vždy nově vytvořené privátní jméno b s vlastním záznamem v globální tabulce jmen. Kdyby bylo porovnání parametrů procesů prováděno na přísněji definovanou shodu jmen, algoritmus by pro daný výraz nebyl konečný.

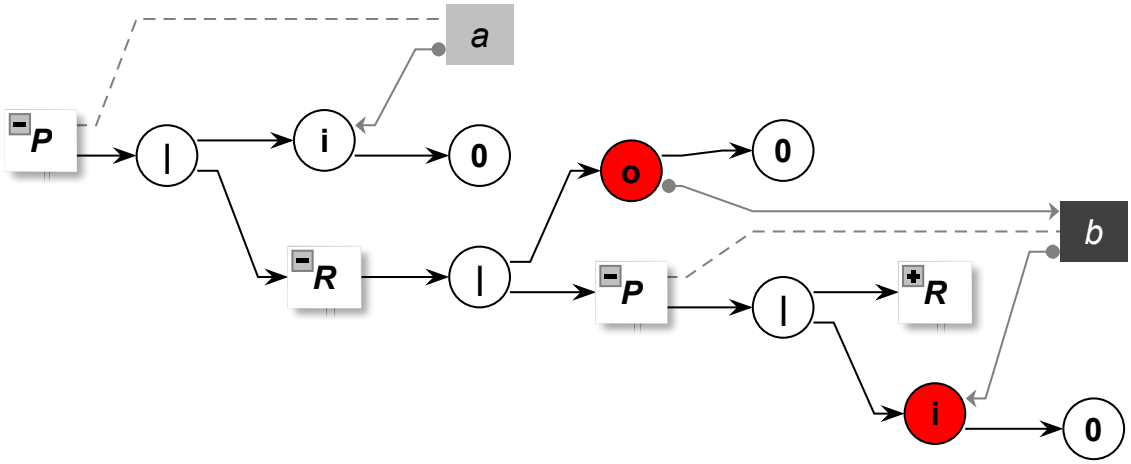
Existuje-li pouze konečný počet definovaných procesů a konečný počet původních jmen, pak je s použitím výše uvedené podmínky algoritmus vyhledávání redukcí konečný. Počet definovaných procesů i původních jmen je nutně konečný, neboť je omezen konečným vstupem od uživatele.

Přejděme nyní k principu generování seznamu redexů. Je-li při průchodu stromem interní reprezentace navštíven uzel nepozorovatelného prefixu, je přidán jako součást nového redexu do výstupního seznamu. Na rozdíl od nepozorovatelného prefixu, navštívené vstupní či výstupní prefixy nemohou být do seznamu přidány okamžitě. Místo toho je při jejich navštívení vytvořen pomocný seznam akcí, do něhož je daný prefix uložen. Tento seznam akcí je pak propagován strukturou stromu směrem nahoru k uzlům paralelní kompozice. Při návštěvě uzlu paralelní kompozice jsou seznamy akcí všech potomků vzájemně porovnány, přičemž jsou extrahovány všechny komplementární dvojice navštívených prefixů. Tyto dvojice prefixů jsou pak uloženy do výstupního seznamu v podobě redexů.

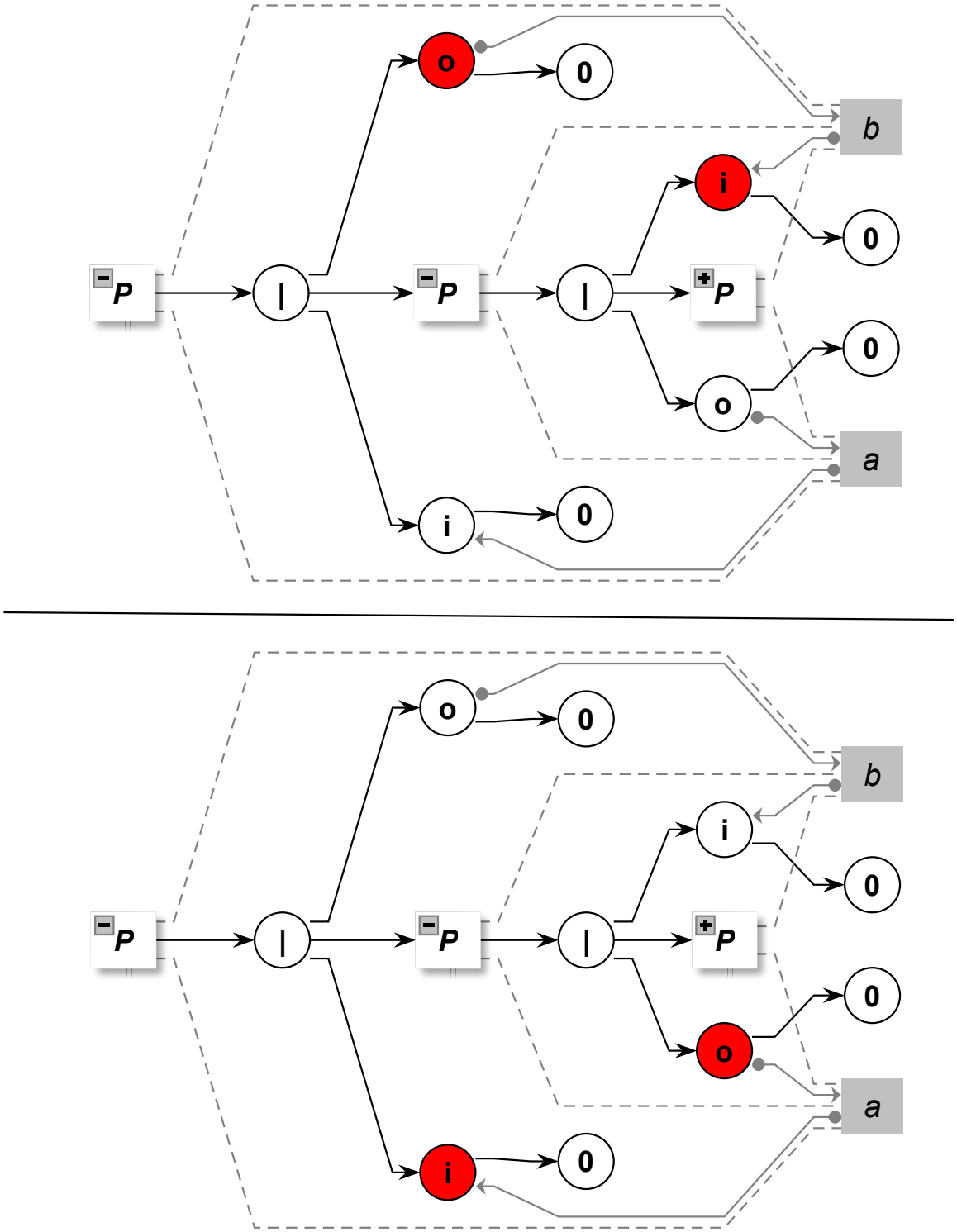
Klíčovýou datovou strukturou používanou v implementaci algoritmu je právě **seznam akcí** implementovaný třídou `ActionList`. Ta obsahuje dva seznamy – jeden pro ukládání vstupních prefixů a druhý pro výstupní prefixy. Seznam akcí je předáván ve směru od listových uzlů ke kořenovému uzlu. Další významnou strukturou, která je předávána v opačném směru, je takzvaný **redukční kontext** implementovaný třídou `ReductionContext`. Ta obsahuje aktuální kontext zpracovávaného uzlu a informace o všech předchozích aplikacích procesů vyskytujících se v prohledávané větvi stromu včetně předaných jmen a kontextu.



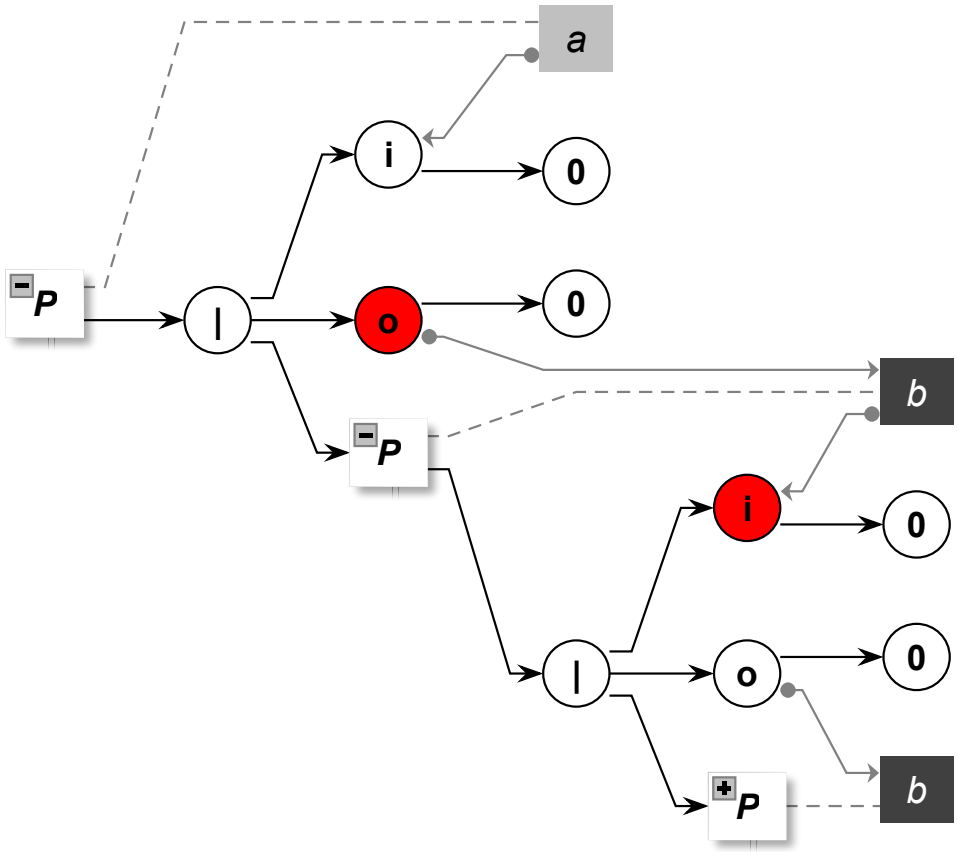
Obrázek 6.7: Grafická reprezentace nekonečného výrazu $\bar{a}.0 \mid P\langle a\rangle$, kde $P(a) \stackrel{\text{def}}{=} a.0 \mid P\langle a\rangle$. Nekonečný rozvoj výrazu je způsoben přítomností aplikace porocesu P v definici daného procesu. V grafické reprezentaci jsou barevně zvýrazněny redukovatelné uzly vizualizované části výrazu.



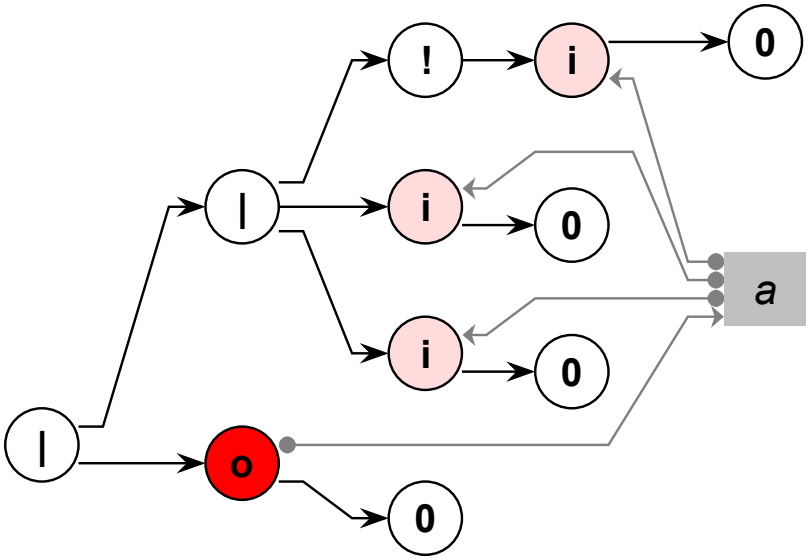
Obrázek 6.8: Grafická reprezentace nekonečného výrazu $P\langle a\rangle$, kde $P(a) \stackrel{\text{def}}{=} (a.0 \mid R)$ a $R \stackrel{\text{def}}{=} (\nu b)(\bar{b}.0 \mid P\langle b\rangle)$. Jedná se o nepřímou rekurzi, kdy aplikace procesu R je součástí definice procesu P a naopak. Červené uzly označují proveditelný redex nalezený algoritmem pro vyhledávání redukcí.



Obrázek 6.9: Grafická reprezentace dvojice vstupně výstupních redexů nalezených algoritmem pro vyhledávání redukcí ve výrazu $P\langle a, b\rangle$, kde $P(a, b) \stackrel{\text{def}}{=} (\bar{b}.0 \mid a.0 \mid P\langle b, a\rangle)$ Za povšimnutí stojí, že s každou aplikací procesu P se mění pořadí předávaných jmen a a b.



Obrázek 6.10: Grafická reprezentace nekonečného výrazu $P\langle a\rangle$, kde $P(a) \stackrel{\text{def}}{=} (\nu b)(\bar{b}.0 \mid a.0 \mid P\langle b\rangle)$. Definice procesu P je podobná definici procesu P z obrázku 6.9 s tím rozdílem, že jméno b je privátní – je definováno operací restrikce uvnitř procesu P. Červené uzly označují proveditelný redex nalezený algoritmem pro vyhledávání redukcí.



Obrázek 6.11: Grafická reprezentace výrazu $\bar{a}.0 \mid (a.0 \mid a.0 \mid !a.0)$, jehož součástí je operace replikace generující nekonečný počet proveditelných redexů.

Navíc obsahuje také speciální příznak určující, zda mají být extrahované redexy přidávány do výstupního seznamu. To se hodí například při průchodu větví replikační kopie, která ještě nebyla vizualizována.

Operace replikace představovala pro implementaci algoritmu vyhledávání redukcí rovněž nemalý problém. Obdobně jako aplikace procesů, je totiž i operace replikace určena pro modelování potenciálně nekonečného chování systémů. Ve výrazu obsahujícím replikaci tak může existovat nekonečné množství redexů. Příkladem budiž výraz $\bar{a}.0 \mid !a.0$ jehož expandovaná podoba je zachycena na obrázku 6.11.

Za účelem vyřešení daného problému bylo nutné rozšířit interní reprezentaci operace replikace o pomocný atributu, který určuje, zda se jedná o replikační uzel uvozující původní podstrom výrazu či jeho kopii. V případě kopie je dále rozlišováno, zda je daná kopie vizualizována, nebo zda se jedná o pomocnou interní kopii určenou pouze k účelu vyhledávání redukcí. Následníkem každého uzlu paralelní kompozice `ParallelReplicationExpression`, který předchází uzlu replikace, je tak vždy právě jeden **originální uzel replikace**, maximálně jeden **pomocný uzel replikace**, libovolný počet uzlů **vizualizované replikační kopie** vytvořených na žádost uživatele a libovolný počet uzlů různých typů, které vnikly z větve replikační kopie provedením redukce.

Existence pomocného uzlu replikace je podmíněna nepřítomností vizualizované replikační kopie, tedy neobsahuje-li uzel paralelní kompozice předcházející replikaci jako svého následníka žádnou vizualizovanou replikační kopii, pak musí obsahovat pomocný uzel replikace. Tím je zajištěno, že při porovnávání seznamů akcí během vyhledávání redukcí, je každý jedinečný redex, obsažený v podvýrazu uvozeném operací replikace, nalezen alespoň jednou (v uzlu paralelní kompozice je totiž porovnáván seznam akcí z originální replikační větve a pomocné replikační větve). V případě, že následníkem paralelní kompozice je alespoň jedna vizualizovaná replikační kopie, pak je zbytečné procházet i pomocnou replikační větev stromu, neboť veškeré dostupné redexy budou nalezeny při porovnání seznamu akcí pocházejících z originální replikační větve a její vizualizované replikační kopie.

V algoritmu vyhledávání redukcí jsou tak, co se týče operace replikace, vyhledávány redexy pouze ve vizualizovaných replikačních větvích, nebo v původní replikační větvi a odpovídající pomocné replikační větvi. Vždy je tedy prováděno vyhledávání pouze v konečném počtu podstromů, což implikuje konečný počet nalezených redexů.

Kompletní algoritmus vyhledávání redukcí, popsaný v pseudokódu, je k nahlédnutí v příloze B.

6.7 Algoritmus provedení redukce

Úkolem správce grafu je kromě vyhledávání redukcí také jejich provádění. Podmínkou nutnou k provedení redukce je existence kompletního redexu (viz. definice 6.6.6) tvořeného nepozorovatelným prefixem nebo dvojicí vstupního a výstupního prefixu. Provedením redukce dochází ke **zjednodušení** výrazu a v případě vstupně-výstupní redukce také k **navázání předávaných jmen** na parametry vstupního prefixu.

V rámci zjednodušení mohou být z výrazu odstraněny nejen interagující prefixy, jež byly součástí aktivovaného redexu, ale i některé další části výrazu, jak bude popsáno později.

Co se týče navázání předávaných jmen, mohou obecně nastat následující situace:

1. Vzájemně reagující prefixy obsahují pouze společný komunikační kanál, nikoliv předávaná jména či parametry.

2. Vzájemně reagující prefixy obsahují společný komunikační kanál, prostřednictvím něhož dochází k předávání jmen. Tato jména jsou přitom definována jak v podvýrazu výstupního prefixu, tak i v podvýrazu vstupního prefixu.

3. Vzájemně reagující prefixy obsahjí společný komunikační kanál, prostřednictvím něhož dochází k předávání jmen, přičemž však některé z předávaných jmen není definováno v podvýrazu vstupního prefixu. (Tato situace je v anglických publikacích označována slovním spojením *scope extrusion*. V této práci ji budeme označovat jako *únik jména z rozsahu*.) Příkladem může být následující redukce: $(\nu n)(\bar{x}\langle n\rangle.P) \mid Q \mid x(a).R \longrightarrow (\nu n)(P \mid R\{n/a\}) \mid Q$.

4. Vzájemně reagující prefixy obsahjí společný komunikační kanál, prostřednictvím něhož dochází k předávání jmen, přičemž však některé z předávaných jmen je označeno stejným názvem jako některé jiné jméno definované v podvýrazu vstupního prefixu. (Tato situace je v anglických publikacích označována slovním spojením *scope intrusion*. V této práci ji budeme označovat jako *narušení rozsahu jména*.) Příkladem může být následující redukce: $\bar{x}\langle z\rangle.P \mid (\nu z)(x(a).Q \mid R) \longrightarrow P \mid (\nu z')(Q\{'z/z\}\{z/a\} \mid R\{z'/z\})$.

Poznámka. Každé jméno může být v interní reprezentaci vizualizovaného výrazu definováno (ve smyslu vytvoření) pouze na některém z následujících míst: v kořenovém uzlu výrazu (definuje všechna volná jména vyskytující se ve výrazu), v uzlu vstupního prefixu (definuje parametry prefixu) nebo v uzlu restrikce (definuje privátní jména). Z vlastnosti redukce vyžadující, aby žádný z předchůdců interagujících prefixů nebyl blokujícím uzlem (viz. definice 6.6.4), vyplývá, že k úniku jména z rozsahu při provádění redukce může dojít pouze v případě, že je předávané jméno privátní. Pokud by totiž bylo vytvořeno v kořenovém uzlu, pak by zřejmě muselo být definováno (známo) i v podvýrazu vstupního prefixu a k úniku z rozsahu by tedy nedošlo. Stejně tak pokud by bylo vytvořeno jako parametr vstupního prefixu, nebyla by redukce realizovatelná, neboť uzel vstupního prefixu je blokujícím uzlem.

Obdobně jako v předchozí kapitole zavedeme nyní několik pomocných pojmů, které budou následně použity při popisu algoritmu redukce.

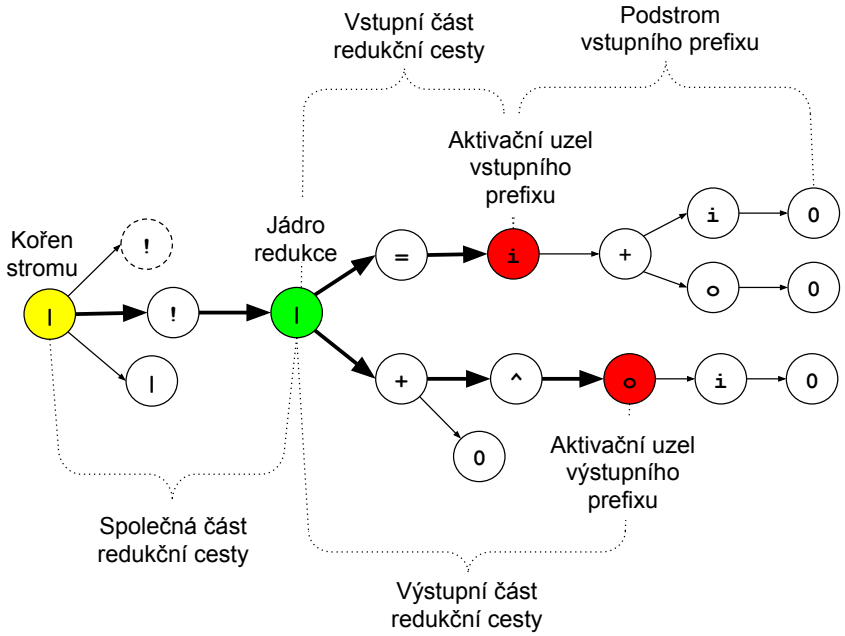
Definice 6.7.1. ***Aktivační uzel*** *redukce je uzel prefixu, jenž byl součástí vybraného redexu.*

Definice 6.7.2. ***Jádrem*** *vstupně-výstupní redukce je uzel paralelní kompozice (`ParallelExpression` nebo `ParallelReplicationExpression`), který je nejbližším společným předchůdcem obou aktivačních uzlů.*

Definice 6.7.3. ***Redukční cesta*** *je cesta vedoucí od kořenového uzlu stromu k aktivačnímu uzlu redukce (včetně). V případě vstupně-výstupní redukce existují právě dvě redukční cesty. Průnik těchto dvou cest budeme označovat jako* ***společnou část redukční cesty****. Sufix redukční cesty od jádra redukce směrem k aktivačnímu uzlu vstupního prefixu budeme označovat jako* ***vstupní část redukční cesty*** *a obdobně sufix redukční cesty od jádra redukce směrem k aktivačnímu uzlu výstupního prefixu budeme označovat jako* ***výstupní část redukční cesty****.*

Definice 6.7.4. ***Podstrom vstupního prefixu*** *je maximální podstrom stromu redukovaného výrazu, jehož kořenovým uzlem je aktivační uzel vstupního prefixu.*

Obrázek 6.12 graficky znázorňuje výše definované pojmy.



Obrázek 6.12: Strom intrerní reprezentace výrazu s tučně vyznačenými redukčními cestami.

Vstupem algoritmu redukce je obousměrně vázaný strom interní reprezentace výrazu, nad nímž má být provedena redukce a kompletní redex, který redukci vyvolal. Výstupem je poté upravený strom téhož výrazu.

Algoritmus redukce je založen na průchodu redukční cestou stromu ve směru od kořenového uzlu k aktivačním uzlům, kdy při návštěvě uzlů konkrétních typů jsou prováděny následující úpravy:

- Úprava konkretizace: Každý navštívený **uzel konkretizace** je označen jako redukovaný. To má význam pro vytváření redukčního kontextu v algoritmu vyhledávání redukcí.

- Úprava sumy: Každý navštívený **uzel sumy** je ze stromu odstraněn a k jeho nejbližšímu předchůdci je připojen jako náhrada právě ten z jeho následníků, který je jako jediný součástí redukční cesty. Tato úprava odpovídá výběru konkrétní alternativy a odstranění všech ostatních z operace sčítání.

- Úprava replikace: Pro každý navštívený **uzel replikace** R, je nejprve vytvořena jeho záložní kopie R'. Následně je daný uzel replikace z výrazu odstraněn a k jeho předchůdci, kterým je zřejmě uzel replikační paralelní kompozice, je připojen jeho následník. Důvodem této úpravy je to, že provedení redukce mění strukturu podstromu, a proto již daný uzel nemůže být součástí replikačního rozvoje, který se vyznačuje tím, že všechny podvýrazy operace replikace mají stejnou strukturu. V případě, že byl odstraněný uzel replikace replikačním originálem, pak jeho funkci převezme vytvořená záložní kopie R', která je tímto připojena k jeho předchůdci. Nastane-li odstraněním daného uzlu situace, kdy žádný z následníků replikační paralelní kompozice není uzlem replikační kopie, pak vytvořený záložní uzel R' převezme funkci pomocného replikačního uzlu. Přítomnost tohoto uzlu jako následníka replikační paralelní kompozice je podstatná po algoritmus vyhledávání redukcí.

- Úprava aktivačního uzlu: Každý navštívený **aktivační uzel** je ze stromu odstraněn a k jeho nejbližšímu předchůdci je připojen jako náhrada jeho následník.

Výše uvedené úpravy jsou aplikovatelné jak na redukci nepozorovatelného prefixu, tak i na vstupně-výstupní redukci. Algoritmus vstupně-výstupní redukce musí navíc pokrývat všechny možné případy navázání jmen vyjmenované dříve v této kapitole, z nichž poslední tři mají všechny za následek změnu referencí jmen. V případě úniku jména z rozsahu pak dochází dokonce i ke změně struktury stromu v důsledku migrace některých uzlů restrikce z původní pozice uvnitř výstupní části redukční cesty směrem k redukčnímu jádru.

Následující kroky algoritmu vstupně-výstupní redukce, které zajišťují správné navázání a substituci jmen, jsou prováděny, obdobně jako předchozí úpravy, v rámci jediného průchodu redukční cestou stromu a v rámci navazujícího průchodu podstromu vstupního prefixu:

- V rámci inicializace algoritmu vstupně-výstupní redukce je ještě před samotným průchodem redukční cesty vytvořen pomocný **seznam privátních jmen** $PrivateList$, který je naplněn referencemi všech privátních jmen, které jsou součástí seznamu předávaných jmen aktivačního uzlu výstupního prefixu.

- Pro každý navštívený **uzel restrikce** R nacházející se na **společné části redukční cesty**, jsou ze seznamu $PrivateList$ odstraněna všechna jména, která jsou obsažena v seznamu definovaných (omezených) jmen navštíveného uzlu R.

- Při návštěvě **jádra redukce** J, je nejprve ověřeno zda je seznam $PrivateList$ prázdný. Pokud ano, pokračuje se dalším krokem algoritmu, v opačném případě zřejmě došlo k úniku jména z rozsahu a proto musí být provedeny následující akce:

  - Všechna jména, která zbyla v seznamu $PrivateList$ jsou označena speciálním příznakem přejmenování, který zajistí, že při výpisu textové reprezentace, bude odpovídající jméno zapsáno ve formátu $\langle nazev\_jmena \rangle \# \langle NumID \rangle$, čímž bude

jednoznačně rozlišitelné od všech dalších jmen stejného názvu (pokud se taková ve výrazu vyskytují).

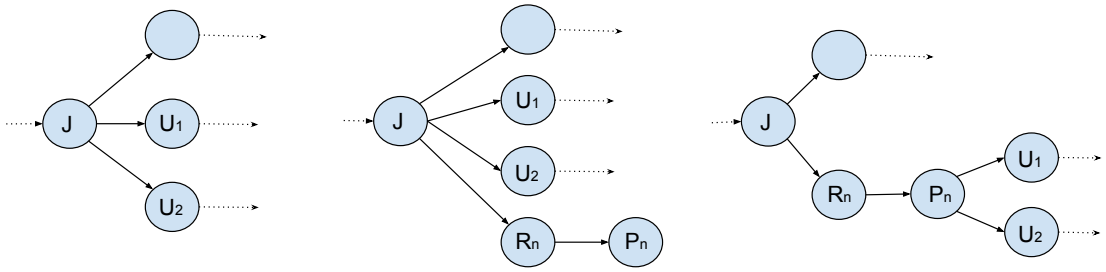
  - Pokud je jádro redukce uzlem replikační paralelní kompozice (`ParallelReplicationExpression`), provede se nejprve kontrola uzlů U_1 a U_2, pro které platí, že jsou přímými následníky jádra redukce nacházejícími se na vstupní nebo výstupní části redukční cesty. Smyslem této kontroly je ověřit, zda některý z těchto uzlů U_i není replikačním uzlem a pokud ano, aplikovat na daný uzel úpravu replikace, jež byla popsaná výše.

    Po provedení kontroly a případných úprav obou uzlů je vytvořen nový uzel restrikce R_n, jehož seznam omezených jmen je inicializován přidáním všech jmen, která zbyla v seznamu $PrivateList$. Tento uzel je připojen do struktury stromu jako následník jádra redukce. Jeho následníkem je pak další nově vytvořený uzel obyčejné paralelní kompozice P_n, k němuž jsou přepojeny uzly $U1$ a $U2$. Tento princip navázání uzlů je demonstrován na obrázku 6.13.
  - Pokud je jádro redukce uzlem obyčejné paralelní kompozice (`ParallelExpression`) a má více než 2 následníky je provedena výše popsaná modifikace spočívající v připojení nového uzlu restrikce R_n za jádro aplikace. V případě, že má jádro pouze 2 následníky, je rovněž vytvořen nový uzel restrikce, který je však připojen do struktury stromu jako nejbližší předchůdce jádra. Jeho přímým předchůdcem se přitom stává původní přímý předchůdce jádra.

- Pro každý navštívený **uzel restrikce** R_o, který je součástí **výstupní části redukční cesty**, jsou ze seznamu definovaných (omezených) jmen navštíveného uzlu R_o odebrána všechna jména která jsou obsažena v seznamu $PrivateList$. (Tato jména jsou totiž v důsledku předchozího kroku již definována výše ve struktuře stromu.) V případě, že po odstranění těchto jmen je seznam uzlu R_o prázdný, pak je celý uzel restrikce R_o zbytečný a je tedy ze stromu odstraněn, přičemž jeho nejbližší předchůdce P a nejbližší následník N jsou spojeni novou hranou (P,N). Dále v případě, že uzel P je uzlem paralelní kompozice libovolného typu (`ParallelExpression` nebo `ParallelReplicationExpression`), ověří se, zda není uzel N rovněž uzlem obyčejné paralelní kompozice (`ParallelExpression`). Pokud ano, jsou oba uzly sjednoceny tím způsobem, že všichni následníci uzlu N jsou připojeni k uzlu P a uzel N je ze stromu odstraněn.

- Pro každý navštívený uzel **uzel restrikce** R_i, který je součástí **vstupní části redukční cesty**, je každé jméno n ze seznamu definovaných (omezených) jmen navštíveného uzlu R_i porovnáno na shodu názvu s libovolným jménem obsaženým v seznamu předávaných jmen aktivačního uzlu výstupního prefixu. Pokud se prokáže shoda, je jméno n označeno příznakem přejmenování, jehož význam byl nastíněn výše.

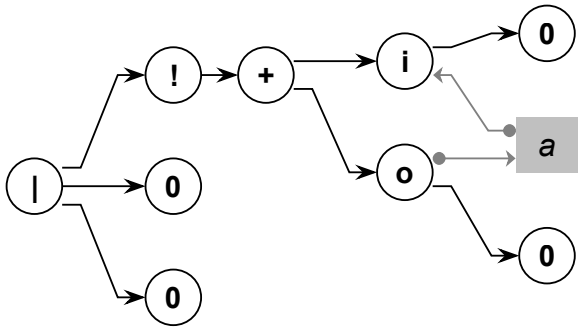
- Poslední krok algoritmu spočívá v **průchodu podstromu vstupního prefixu**, jehož smyslem je nalézt a následně nahradit všechny výskyty reference jména, které bylo definováno aktivačním uzlem vstupního prefixu, referencí na odpovídající jméno obsažené v seznamu aktivačního uzlu výstupního prefixu. Navíc v každém navštíveném uzlu restrikce a vstupního prefixu je pro každé definované jméno n provedeno porovnání na shodu názvu s libovolným jménem ze seznamu aktivačního uzlu výstupního prefixu. Pokud je shoda nalezena, je dané jméno n označeno příznakem přejmenování.



Obrázek 6.13: Modifikace interní reprezentace výrazu spočívající v navázání nového uzlu restrikce a paralelní kompozice při úniku jména z rozsahu.

6.8 Algoritmus zjednodušení výrazu

Přestože požadavek na zjednodušování výrazů nebyl součástí zadání diplomové práce ani původního návrhu aplikace, ukázal se být žádoucím v důsledku implementace algoritmu pro provádění redukcí. Zejména u výrazů obsahujících operaci replikace totiž může opakovaným prováděním redukcí docházet k nárůstu počtu uzlů typu Nil, jako potomků operace paralelní kompozice (viz. obrázek 6.14). Tyto uzly však nemají žádný vliv na výpočetní potenciál daného výrazu a navíc spíše znepřehledňují textovou i grafickou reprezentaci výrazu.

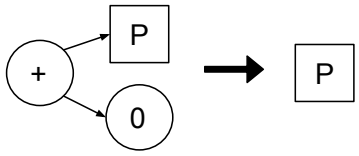
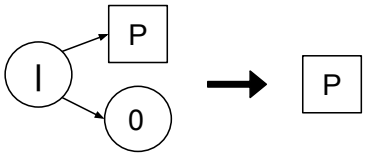
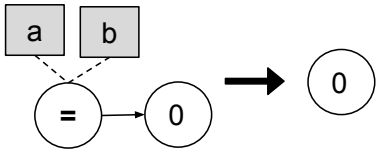
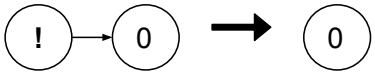


Obrázek 6.14: Grafická reprezentace výrazu $!(\bar{a}.0 + a.0)$ po provedení jedné redukce.

Dalším podnětem pro implementaci algoritmu zjednodušení bylo rozšíření funkcionality aplikace o manuální vytváření neomezeného počtu replikačních kopií (kliknutím pravého tlačítka myši na uzel replikace a výběrem položky z kontextového menu). Opačný proces, tedy manuální odstranění replikačních kopií, však nebyl v současné verzi aplikace implementován. Jako alternativní řešení byl i proto navržen algoritmus zjednodušení, který dokáže, mimo jiné, automaticky odstranit z výrazu všechny replikační kopie.

Teoretickým základem algoritmu zjednodušení je množina axiomů a pravidel definovaných procesní algebrou π-kalkul. Tyto axiomy mohou být rozděleny do čtyř kategorií podle toho, jaký efekt má jejich aplikace na interní reprezentaci výrazu.

První skupinu 6.1 tvoří axiomy a pravidla popisující vlastnosti procesu Nil. Aplikací těchto pravidel je možné ve stromu interní reprezentace výrazu nahradit všechny podstromy, jejichž součástí není uzel konkretizace procesu ani žádný z uzlů vstupního, výstupního nebo nepozorovatelného prefixu, uzlem typu Nil. První dva axiomy rovněž umožňují odstranit všechny přímé následníky typu Nil libovolného uzlu sumy či paralelní kompozice.

`P + 0` → `P`	
`P | 0` → `P`	
`(^z)0` → `0`	
`[a = b]0` → `0`	
`!0` → `0`	

Tabulka 6.3: Změny textové reprezentace (levý sloupec) a grafické reprezentace (pravý sloupce) při aplikaci pravidel 6.1 v algoritmu zjednodušení.

$$\begin{aligned} P + 0 &= P \\ P \mid 0 &= P \\ \nu z 0 &= 0 \\ [a = b]\ 0 &= 0 \\ !\ 0 &= 0 \end{aligned} \tag{6.1}$$

Tabulka 6.3 znázorňuje, k jakým změnám dojde v textové a grafické reprezentaci aplikováním výše uvedených pravidel 6.1.

Další kategorii tvoří pravidlo 6.2 vycházející z definice operace replikace. Aplikací tohoto pravidla je možné odstranit ze stromu výrazu všechny podstromy reprezentující replikační kopie.

$$P|!P = !P \tag{6.2}$$

Pravidlo 6.3 vychází z definice prefixu shody. Aplikací tohoto pravidla jsou ze stromu výrazů odstraněny všechny uzly prefixu shody, u nichž jsou porovnávaná jména shodná. Je zřejmé, že přítomnost těchto platných prefixů ve výrazu je zbytečná.

$$[a = a]\ P = P \tag{6.3}$$

Poslední skupina axiomů a pravidel 6.4 umožňuje sjednotit v interní reprezentaci výrazu některé po sobě následující uzly stejného typu. Toto platí výhradně pro uzly sumy, paralelní kompozice, replikace a restrikce, nikoliv však pro uzly prefixů.

$$\begin{gathered}(P + Q) + R = P + (Q + R) = P + Q + R \\ (P \mid Q) \mid R = P \mid (Q \mid R) = P \mid Q \mid R \\ !!P = !P \\ (\nu\ r)(\nu\ s)P = (\nu\ r, s)P\end{gathered} \tag{6.4}$$

Tabulka 6.4 znázorňuje, k jakým změnám dojde v textové a grafické reprezentaci aplikováním výše uvedených pravidel 6.2, 6.3 a 6.4.

Následující výčet shrnuje všechny úpravy, které jsou provedeny nad interní reprezentací výrazu při použití algoritmu zjednodušení:

- nahrazení výpočetně nerelevantních podstromů, uzlem typu Nil
- odstranění následníků typu Nil z uzlů sumy a paralelní kompozice
- odstranění replikačních kopií
- odstranění platných prefixů shody
- sjednocení za sebou následujících uzlů sumy, paralelní kompozice, replikace a restrikce

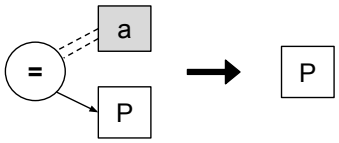
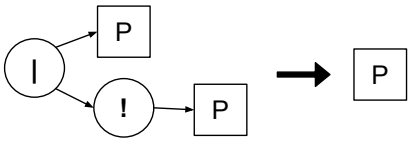
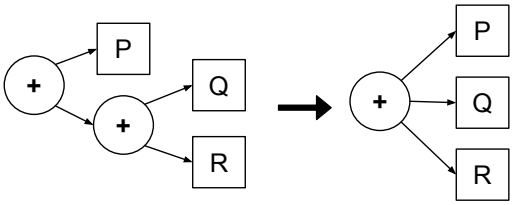
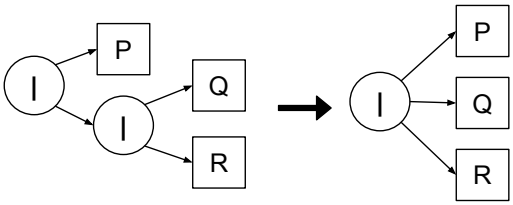
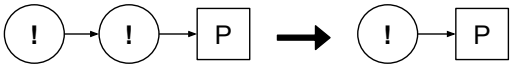
Zjednodušení výrazů v aplikaci zajišťuje správce zjednodušení implementovaný třídou `SimplifierManager`. Vstupem algoritmu je strom interní reprezentace aktuálně vizualizovaného výrazu. Výstupem je poté zjednodušený strom téhož výrazu.

Princip algoritmu je založen na rekurzivním průchodu stromem výrazu, kdy jsou nejprve přímým průchodem nalezeny všechny listové uzly typu Nil a neinstanciované konkretizace procesu. Od nich je pak postupováno strukturou stromu směrem ke kořenovému uzlu v důsledku návratu z rekurzivního volání. Při tomto průchodu směrem vzhůru je v každém navštíveném uzlu U, který není uzlem vstupního, výstupního nebo nepozorovatelného prfixu, provedena analýza jeho následníků

Je-li uzel U uzlem sumy, nebo uzlem obyčejné paralelní kompozice, pak jsou odstraněny ze seznamu následníků všechny uzly typu Nil. Pokud tímto nastane situace, že má uzel U již jen jednoho nebo žádného následníka, je sám ze stromu odstraněn a k uzlu jeho přímého předchůdce je připojen jeho jediný zbývající následník nebo nový uzel typu Nil.

Zvláštní přístup v tomto ohledu vyžaduje uzel replikační paralelní kompozice. Při procesu odstraňování následníků jsou odstraněny nejen uzly typu Nil, ale také uzly replikační kopie. Samotný uzel U pak může být odstraněn z důvodu nedostatku následníků pouze pokud jeho následníkem není uzel původní replikace.

Je-li uzel U uzlem replikace, restrikce, prefixu shody, konkretizace procesu nebo abstrakce procesu a je-li jeho následníkem uzel typu Nil, pak je uzel U odstraněn ze struktury stromu a jeho přímý předchůdce je spojen s daným následníkem. K odstranění uzlu U dojde rovněž v případě, že se jedná o uzel platného prefixu shody.

`[a = a]P` → `P`	
`P | !P` → `P`	
`P + (Q + R)` → `P + Q + R`	
`P | (Q | R)` → `P | Q | R`	
`!!P` → `!P`	
`(^r)(^s)P` → `(^r,s)P`	

Tabulka 6.4: Změny textové reprezentace (levý sloupec) a grafické reprezentace (pravý sloupce) při aplikaci pravidel 6.2, 6.3 a 6.4 v algoritmu zjednodušení.

Kromě analýzy následníků je u uzlů sumy, paralelní kompozice a restrikce provedena rovněž analýza uzlů předchůdce. Je-li předchůdcem uzel stejného typu, pak jsou následnící navštíveného uzlu U svěřeny tomuto předchůdci a uzel U je odstraněn. Speciální přístup je opět vyžadován v případě replikační paralelní kompozice. Je-li předchůdcem uzel paralelní kompozice, pak jsou mu předáni pouze následníci, kteří nejsou uzlem replikace. Obdobně pak, je-li předchůdcem uzel původní replikace R, jsou tito následníci svěřeny předchůdci uzlu R, kterým je zřejmě replikační paralelní kompozice. Algoritmus musí rovněž zajistit, že po provedení všech úprav bude obnoven pomocný replikační uzel, jehož existence je nezbytná pro algoritmus vyhledávání redukcí.

6.9 Komponenty uživatelského rozhraní

Grafické uživatelské rozhraní aplikace bylo implementováno s využitím knihovny **Swing**, která nabízí prostředky pro tvorbu různých prvků uživatelského rozhraní – oken, dialogů, tlačítek, seznamů a dalších. Tyto prvky jsou ve vytvořené aplikaci uspořádány do hierarchické struktury, v níž je nejvýše postaveným prvkem hlavní okno aplikace, implementované třídou `MainFrame`. Toto okno sdružuje všechny komponenty uživatelského rozhraní, prostřednictvím kterých je pak uživateli umožněno aplikaci ovládat a prohlížet výsledky vykonaných akcí. Kromě hlavního okna, je součástí grafického rozhraní aplikace ještě pomocné okno nápovědy implementované třídou `HelpFrame` a okno s informacemi o aplikaci `AboutFrame`.

Mezi klíčové komponenty uživatelského rozhraní vytvořené aplikace patří **plátno grafu**, zobrazující grafický výstup aplikace, a **textová konsole**, jež je hlavním prostředkem pro textový vstup aplikace.

Správu obsahu plátna grafu a sběr informací o manipulaci s grafickými entitami má na starost správce grafu, jehož implementační detaily byly rozebrány v kapitole 6.4.

Přestože textová konsole nevypadá na první pohled složitě, docílit toho, aby při použití působila dojmem reálného textového terminálu používaného například v unixových systémech, nebylo snadným úkolem. Pro implementaci interní textové oblasti konsole byla použita třída `JTextArea` z knihovny Swing, která reprezentuje obecnou komponentu určenou pro víceřádkový textový vstup. Tato komponenta může být nakonfigurována tak, že umožňuje buď editaci celého textu, nebo naopak není upravovatelná vůbec. Textový terminál je však zpravidla tvořen dvěma částmi – editovatelným příkazovým řádkem, který umožňuje uživateli zadávat příkazy, a neupravovatelnou částí zobrazující odeslané příkazy. Součástí příkazového řádku je navíc nemodifikovatelný prompt, který indikuje připravenost terminálu přijímat příkazy.

Pro dosažení podobné funkcionality u objektu třídy `JTextArea`, bylo nutné vytvořit vlastní navigační filtr (`NavigationFilter`) a dokumentový filtr (`DocumentFilter`). Za účelem vyhodnocování stisku klávesy ENTER a zpracování vždy pouze relevantní části textu, která odpovídá poslednímu zadanému příkazu, bylo potřeba vhodným způsobem implementovat rozhraní `DocumentListener`. Právě díky tomu je konsole schopná korektním způsobem zpracovávat nejen jednořádkový vstup, ale i dávkový vstup vložený prostřednictvím schránky.

Jako součást konsole byla implementována také **historie příkazů**, kterou může uživatel procházet pomocí klávesových šipek a dokonce dočasně modifikovat její položky. (Při odeslání příkazu je obnoven původní stav přepsané položky.)

Dalšími komponentami, které tvoří významnou část hlavního okna aplikace, je **menu** a **panel rychlého přístupu**. Tyto komponenty obsahují vzájemně synchronizované prvky

(položky menu a tlačítka) sloužící pro ovládání aplikace. Pro pohodlnější ovládání aplikace bez použití myši, byla u významných položek z menu implementována podpora klávesových zkratek.

Komponentou, která představuje textový výstup aplikace, je **panel pro zobrazení textové reprezentace** vizualizovaného výrazu a **editor definovaných procesů**. Obě komponenty používají pro implementaci interní textové oblasti třídu `JTextPane`, která podporuje formátování textu. Konkrétní úseky textu jsou formátovány (zvýrazněny určitou barvou) dle příznaků třídy `TextBlob`. Princip vytváření seznamu těchto objektů byl popsán v kapitole 6.3.

Poslední, dosud nezmíněnou komponentou hlavního okna aplikace, je **panel redukcí** obsahující seznam (tabulku) všech redexů vizualizovaného výrazu. Tento panel je v počátečním nastavení skryt za levým okrajem okna, aby nezabíral zbytečně místo, pokud by uživatel neměl zájem o jeho použití. Je možné jej však kdykoliv zobrazit a případně opět skrýt použitím odpovídající položky v menu nebo textovým příkazem. Zvýraznění položky v seznamu redukcí je synchronizováno s výběrem odpovídajících uzlů v grafu. Toho bylo, mimo jiné, docíleno vytvořením modelu tabulky (`ReductionTableModel`), v němž je uložen skutečný seznam redukcí (nejen jeho textová podoba) obsahující objekty interní reprezentace výrazu.

Kapitola 7

Závěr

V této práci byly nejprve objasněny základní pojmy a principy procesních algeber a představena trojice procesních algeber CSP, CCS a ACP. Rovněž byly popsány obvyklé způsoby grafického vyjádření výrazů. Samostatná kapitola pak byla věnována procesní algebře π-kalkul. V této kapitole byla, kromě syntaxe a sémantiky výrazů, definována také relace redukce a relace přechodu. Následně bylo na dvou příkladech ukázáno možné použití výrazů π-kalkulu pro popis systémů tvořených vzájemně komunikujícími komponentami.

Další část práce byla věnována analýze existujících aplikací využívajících π-kalkul a srovnání různých textových i grafických reprezentací. Při hledání různých existujících řešení bylo objeveno, že dosud nebyl vytvořen žádný nástroj, který by pracoval s grafickou reprezentací výrazů. Jednalo se vždy pouze o teoretické modely grafických reprezentací.

S ohledem na prozkoumané existující nástroje a grafické modely byla navržena konkrétní textová a grafická reprezentace výrazů π-kalkulu. Při návrhu textové reprezentace bylo snahou zachovat kompatibilitu s textovými reprezentacemi používanými v nástrojích MWB a ABC. Grafická reprezentace pak byla částečně inspirována modelem π-graphs a přizpůsobena pro použití v interaktivní aplikaci. Toto přizpůsobení spočívalo zejména v zavedení dvou různých stavů aplikace procesu a dvou různých způsobů navázání výrazu daného procesu.

Dále byla navržena interní reprezentace výrazů určená pro ukládání a strojové zpracování výrazů uvnitř aplikace. Při návrhu této reprezentace bylo nutné vyřešit problém potenciálně nekonečného rozvoje výrazu způsobeného například kruhovou závislostí aplikovaných procesů nebo existencí operace replikace uvnitř výrazu.

Na základě provedeného návrhu byla následně aplikace implementována. Proces implementace byl zdokumentován v poslední části této práce, která popisuje jak způsob převodu výrazů mezi jednotlivými reprezentacemi, tak i algoritmy použité pro vyhledávání redukcí, provádění redukcí a zjednodušování výrazů.

Vytvořená aplikace, která je, společně s touto technickou zprávou, výstupem diplomové práce, představuje plně funkční nástroj určený pro grafickou vizualizaci výrazů π-kalkulu. Kromě vizualizace poskytuje navíc i podporu pro vyhledávání proveditelných redukcí, jejich provádění a pro automatické zjednodušování vizualizovaných výrazů. Vedle toho nabízí ještě celou řadu dalších funkcí, jako například ukládání textové reprezentace definovaných procesů do textového souboru nebo export grafické reprezentace do souborů různých grafických formátů. Součástí vytvořené aplikace je grafické uživatelské rozhraní, prostřednictvím něhož může uživatel aplikaci ovládat, zadávat výrazy v textové podobě nebo provádět drobné upravy nad grafickou reprezentací výrazů (například rozbalením operace replikace).

Aplikaci by bylo vhodné dále rozšířit o převod výrazů z grafické reprezentace na textovou formou grafického editoru, který by uživatelům umožnil vytvářet a libovolně upravovat graf výrazu. Dalším možným rozšířením, které by vedlo ke zlepšení uživatelské přívětivosti aplikace, by bylo přidání funkce *zpět*, která by po provedení redukce, zjednodušení výrazu či rozbalení replikace, umožňovala návrat k předchozímu stavu výrazu. V kombinaci s komplementární funkcí *vpřed* by tak umožňovala procházet jednotlivé snímky zachycující vývoj grafické reprezentace v důsledku provádění operací.

Literatura

[1] Abadi, M.; Gordon, A.: A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, ročník 148, č. 1, 1999: s. 1–70, ISSN 0890-5401, [Online; navštíveno 10.01.2017].
URL http://www.sciencedirect.com/science/article/pii/S0890540198927407

[2] Bergstra, J. A.; Klop, J. W.: *Fixed Point Semantics in Process Algebras*. Technická zpráva, IW 206, Centre for Mathematics and Computer Science, Amsterdam, 1982.

[3] Bergstra, J. A.; Klop, J. W.: *ACP - A Universal Axiom System for Process Specification*. *CWI Newsletter*, ročník 15, 1987: s. 3–23, ISSN 0168-826X, [Online; navštíveno 10.01.2017].
URL http://oai.cwi.nl/oai/asset/1714/1714A.pdf

[4] Borgström, J.; Gordon, A. D.; Phillips, A.: A Chart Semantics for the Pi-Calculus. *Electronic Notes in Theoretical Computer Science*, ročník 194, č. 2, 2008: s. 3–29, ISSN 1571-0661, [Online; navštíveno 10.01.2017].
URL http://www.sciencedirect.com/science/article/pii/S1571066108000030

[5] Diezmann, C.: A Study of the Importance of Visual Representation and Visual Learning. In *7th International Conference on Thinking*, National Institute of Education, Nanyang Technological University, Singapore, 1997.

[6] Gadducci, F.: Graph rewriting for the π-calculus. *Mathematical Structures in Computer Science*, ročník 17, č. 3, 2007: s. 407–437, ISSN 0960-1295.

[7] Hoare, C. A. R.: *Communicating Sequential Processes* . Prentice Hall International, 1985, ISBN 0-13-153271-5.

[8] J.C.M.Baeten: *A Brief History of Process Algera*. *Theoretical Computer Science*, ročník 335, č. 2-3, 2005: s. 131–146, ISSN 0304-3975, [Online; navštíveno 10.01.2017].
URL http://dx.doi.org/10.1016/j.tcs.2004.07.036

[9] Li, L.: Implementing the π-Calculus in Java. *Journal of Object Technology*, ročník 4, č. 2, 2005: s. 157–177, ISSN 1660-1769, [Online; navštíveno 10.01.2017].
URL http://www.jot.fm/issues/issue$_2$005$_0$3/article5.pdf

[10] Milner, R.: *A Calculus of Communicating Systems*. Technická zpráva, LFCS Report Series ECS-LFCS-86-7, 1986.

[11] Milner, R.: *Action Structures*. Research report, LFCS-92-249, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992, [Online; navštíveno 10.01.2017].

URL http://www.lfcs.inf.ed.ac.uk/reports/92/ECS-LFCS-92-249/ECS-LFCS-92-249.ps

[12] Milner, R.: *Pi-nets: A graphical form of π-calculus*. In *ESOP: Lecture Notes in Computer Science*, ročník 788, editace D. Sannella, Springer Berlin Heidelberg, 1994, ISBN 978-3-540-48376-2, s. 26–42, [Online; navštíveno 10.01.2017].
URL http://dx.doi.org/10.1007/3-540-57880-3$_2$

[13] Milner, R.: *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999, ISBN 0-521-65869-1.

[14] Parrow, J.: Interaction Diagrams. *Nordic Journal of Computing*, ročník 2, č. 4, 1995: s. 407–443, ISSN 1236-6064, [Online; navštíveno 10.01.2017].
URL http://user.it.uu.se/~joachim/interdia.ps.gz

[15] Peschanski, F.; Bialkiewicz, J.-A.: *Modelling and verifying Mobile Systems Using π-Graphs*. In *SOFSEM, Lecture Notes in Computer Science*, ročník 5404, editace Nielsen; aj., Springer Berlin Heidelberg, 2009, ISBN 978-3-540-95891-8, s. 437–448, [Online; navštíveno 10.01.2017].
URL http://dx.doi.org/10.1007/978-3-540-95891-8$_4$0

[16] Peschanski, F.; Klaudel, H.; Devillers, R.: A Decidable Characterization of a Graphical Pi-calculus with Iterators. *Electronic Proceedings in Theoretical Computer Science*, ročník 39, 2010: s. 47–61, ISSN 2075-2180.

[17] Pierce, B. C.: *Programming in the pi-calculus: A tutorial introduction to Pict (Pict Version 4.1)* . 1998, [Online; navštíveno 10.01.2017].
URL
http://www.cs.rpi.edu/academics/courses/spring04/dci/picttutorial.pdf

[18] Rychlý, M.: *Formální specifikace architektur informačních systémů* . FIT VUT v Brně, 2006, [Online; navštíveno 10.01.2017].
URL http://www.fit.vutbr.cz/~rychly/public/docs/formal-architectures/formarch.pdf

[19] Rychlý, M.: *Formal-based Component Model with Support of Mobile Architecture*. Dizertační práce, FIT VUT v Brně, 2009.

[20] Sangiorgi, D.; Walker, D.: *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001, ISBN 0-521-78177-9.

[21] Victor, B.: *The Mobility Workbench User's Guide*. 1995, [Online; navštíveno 10.01.2017].
URL https://www.it.uu.se/profundis/mwb-dist/guide-3.122.pdf

Přílohy

Příloha A

Gramatika pro ANTLRv4

```
grammar PiExpr;


cmds    :   (cmd NL)* cmd;

cmd     :   'agent' ID def      # Agent
        |   'show' sum          # Show
        |   'redlist'           # List
        |   'reduce'            # Reduce
        |   'simplify'          # Simplify
        |   'env' ID?           # Env
        |   'clear'             # Clear
        |   'reset'             # Reset
        |   'help'              # Help
        |   'exit'              # Exit
        |   'quit'              # Exit
        |                       # Empty
        ;

def     :   '=' sum
        |   '(' nlist ')' '=' sum
        |   '=' '(' '\\' nlist ')' sum
        ;

sum     :   par                         # Continuesum
        |   par ('+' par)+              # Summation
        ;

par     :   proc                        # Continuepar
        |   proc ('|' proc)+            # Parallel
        ;

proc    :   NIL                         # Nil
        |   ID '<' nlist '>'            # Concretization
```

```
        |   ID '(' nlist ')'              # Concretization
        |   ID                           # Concretization
        |   pi proc                      # Prefix
        |   '[' NAME '=' NAME ']' proc   # Match
        |   '!' proc                     # Replication
        |   '(' '^' nlist ')' proc       # Restriction
        |   '(' sum ')'                  # Parentheses
        ;

pi      :   TAU '.'                      # Tau
        |   NAME '.' '(' '\\' nlist ')'  # Input
        |   NAME '(' nlist ')' '.'       # Input
        |   NAME '.'                     # Input
        |   '\'' NAME '.' '[' nlist ']'  # Output
        |   '\'' NAME '<' nlist '>' '.'  # Output
        |   '\'' NAME '.'                # Output
        ;

nlist   :  NAME (',' NAME)*
           ;

TAU:    't';
NIL:    '0';
ID:     [A-Z]([a-zA-Z0-9] | '_' | '-')*;
NAME:   [a-z]([a-zA-Z0-9] | '_' | '-')*;

NL:     '\r'? '\n';
WS:     [ \t]+ -> skip;
```

Příloha B

Algoritmus vyhledávání redukcí

Algoritmus 1 Generování seznamu redukcí

```
1: procedure GENERATE-REDUCTION-LIST(Root)
2:     RedCtx ← empty reduction context
3:     RedList ← empty reduction list
4:     if Root ≠ NULL then
5:         VISIT-NODE(Root, RedCtx, RedList)
6:     end if
7:     return RedList;
8: end procedure
```

Algoritmus 2 Generování seznamů akcí průchodem stromu interní reprezentace (1. část)

```
1: function VISIT-NODE(Node, RedCtx, RedList) : ActionList
2:     ActList ← empty action list
3:     if Node ≠ NULL then
4:         switch type of Node do
5:             case Nil :
6:                 break                                   ▷ do nothing
7:             case TauPrefix :
8:                 if RedCtx permits adding into reduction list then
9:                     RedList.add(Node)
10:                end if
11:                break
12:            case InPrefix | OutPrefix :
13:                ActList.add(Node)
14:                break
15:            case MatchPrefix :
16:                if Node is valid then
17:                    SuccNode ← Node.getSuccessor()
18:                    ActList ← VISIT-NODE(SuccNode, RedCtx, RedList)
19:                end if
20:                break
```

Algoritmus 3 Generování seznamů akcí průchodem stromu interní reprezentace (2. část)

```
21:         case ParallelComposition :
22:             ActLists ← empty list of action lists
23:             for each SuccNode ∈ Node.getSuccessors() do
24:                 TmpList ← Visit-Node(SuccNode, RedCtx, RedList)
25:                 ActList.addAll(TmpList)
26:                 ActLists.add(TmpList)
27:                 if RedCtx permits adding into reduction list then
28:                     Extract-Reductions(ActLists, RedList)
29:                 end if
30:             end for
31:             break
32:         case ParallelReplicationComposition :
33:             ActLists ← empty list of action lists
34:             HelperNode ← NULL
35:             OrigActList ← empty action list
36:             for each SuccNode ∈ Node.getSuccessors() do
37:                 if SuccNode is replication helper then
38:                     HelperNode ← SuccNode
39:                     continue
40:                 end if
41:                 TmpList ← Visit-Node(SuccNode, RedCtx, RedList)
42:                 ActList.addAll(TmpList)
43:                 ActLists.add(TmpList)
44:                 if SuccNode is replication original then
45:                     OrigActList ← TmpList
46:                 end if
47:             end for
48:             if RedCtx permits adding into reduction list then
49:                 Extract-Reductions(ActLists, RedList)
50:                 if no replication copy was traversed then
51:                     ActLists ← empty list of action lists
52:                     NewCtx ← copy of RedCtx
53:                     set NewCtx to forbid addition into reduction list
54:                     TmpList ← Visit-Node(HelperNode, NewCtx, RedList)
55:                     ActList.add(TmpList)
56:                     ActList.add(OrigActList)
57:                     Extract-Reductions(ActLists, RedList)
58:                 end if
59:             end if
60:             break
```

Algoritmus 4 Generování seznamů akcí průchodem stromu interní reprezentace (3. část)

```
61:             case Concretization :
62:                 NewCtx ← copy of RedCtx
63:                 SuccNode ← Node.getSucc()
64:                 if SuccNode ≠ NULL then
65:                     if Node was not reduced then
66:                         NewCtx.add(Node.ID, Node.params)
67:                     end if
68:                     ActList ← Visit-Node(SuccNode, NewCtx, RedList)
69:                 else
70:                     if Node is not reused then
71:                         request instantiation of process Node.ID
72:                         SuccNode ← Node.getSucc()
73:                         if SuccNode ≠ NULL then
74:                             NewCtx.add(Node.ID, Node.params)
75:                             ActList ← Visit-Node(SuccNode, NewCtx, RedList)
76:                         end if
77:                     end if
78:                 end if
79:                 break
80:             default :
81:                 for each SuccNode ∈ Node.getSuccessors() do
82:                     ActList ← Visit-Node(SuccNode, RedCtx, RedList)
83:                 end for
84:                 break
85:     end if
86:     return ActList;
87: end function
```

Algoritmus 5 Extrakce vstupně výstupních redexů ze seznamů akcí.

```
1: procedure EXTRACT-REDUCTIONS(ActionLists, RedList)
2:     for i ← 1, ActionLists.size-1 do
3:         ListA ← ActionLists(i)
4:         for j ← i + 1, ActionLists.size do
5:             ListB ← ActionLists(j)
6:             for each InPrefix ∈ ListA.getInputs() do
7:                 for each OutPrefix ∈ ListB.getOutputs() do
8:                     if InPrefix is complementary to OutPrefix then
9:                         RedList.add([InPrefix, OutPrefix])
10:                    end if
11:                end for
12:            end for
13:            for each OutPrefix ∈ ListA.getOutputs() do
14:                for each InPrefix ∈ ListB.getInputs() do
15:                    if InPrefix is complementary to OutPrefix then
16:                        RedList.add([InPrefix, OutPrefix])
17:                    end if
18:                end for
19:            end for
20:        end for
21:    end for
22: end procedure
```

Příloha C

Obsah přiloženého paměťového média

Součástí diplomové práce je CD-ROM, který obsahuje:

- **src.zip** – zdrojový kód vytvořené aplikace
- **pivis.jar** – spustitelný soubor aplikace
- **xproko26.pdf** – elektronická verze technické zprávy
- **latex.zip** – zdrojové kódy k technické zprávě

Příloha D

Návod k instalaci

D.1 Stažení aplikace

Vytvořená aplikace je uložena na CD, které je přiloženo k technické zprávě diplomové práce. Nejaktuálnější verzi aplikace je možné získat z git repozitáře na adrese https://github.com/dprokopo/pi-visualizer.

D.2 Sestavení a spuštění aplikace

V kořenovém adresáři repozitáře se nachází následující soubory:

- **pi-visualizer** – adresář s projektem aplikace
- **yfiles** – obalující modul knihovny yFiles pro práci s grafy
- **LICENSE** – plné znění Apache licence verze 2.0
- **README.md** – pokyny pro sestavení a spuštění aplikace

Sestavení aplikace je možné provést z příkazového řádku pomocí nástroje Maven[1]:

```
$> cd pi-visualizer
$> mvn package
```

Pro spuštění aplikace je možné použít následující příkaz:

```
$> java -jar target/pi-visualizer-1.0-with-dependencies.jar
```

Součástí aplikace je uživatelská příručka zobrazitelná pomocí klávesové zkratky F1.

D.3 Postup pro použití knihovny yFiles

Má-li uživatel k dispozici knihovnu yFiles verze 3.0.0, může ji použít v aplikaci jako náhradu knihovny JGraphX. Knihovna yFiles poskytuje výrazně lepší výsledky při rozmisťování grafických entit na plátně grafu. Pro její aktivaci je nutné provést následující kroky:

[1] https://maven.apache.org/

1. přidat závislost pro danou verzi knihovny do souboru **pom.xml**
2. zkopírovat adresář `yfiles` do adresáře `pi-visualizer/src/main/java/cz/vutbr/fit/xproko26/pivis/gui/graph/graphlib/`
3. v souboru `pi-visualizer/src/main/java/cz/vutbr/fit/xproko26/pivis/gui/graph/GraphManager.java`
4. odkomentovat řádky obsahující text `/***YFILES***/`

Na adrese https://www.yworks.com/products/yfiles-for-java/evaluate je možné zažádat o 60-ti denní evaluační licenci knihovny yFiles.