dipwork.chi (122)

ГОСКОМИТЕТ СССР ПО ДЕЛАМ НАУКИ И ВЫСШЕЙ ШКОЛЫ

НОВОСИБИРСКИЙ ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ

ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

им. Ленинского комсомола

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра вычислительных систем

ТАРАСЮК ИГОРЬ ВАЛЕРЬЕВИЧ

Дипломная работа

ПРИВЕДЕНИЕ ФОРМУЛ АЛГЕБРЫ AFP_2 К КАНОНИЧЕСКОМУ ВИДУ

Научный руководитель

научный сотрудник ИСИ

С. П. Мыльников

Новосибирск 1992

Введение

Для описания параллельных систем и процессов и исследования их поведенческих свойств были предложены различные модели параллельности. В зависимости от представления параллельности они могут быть разбиты на 3 группы.

1. Параллельность-последовательный недетерминизм.

Выполнение двух атомарных действий a и b представляется формулой:

$$a\|b=ab_\vee ba$$

То есть a предшествует b или b предшествует a.

2. Параллельность-одновременность или последовательный недетерминизм.

В этом случае выполнение двух атомарных действий a и b может быть представлено формулой:

$$a\|b=ab_\vee ba_\vee a!b$$

То есть a предшествует b или b предшествует a или a и b выполняются одновременно.

3. Группа параллельных моделей, основанная на концепции процесса, в соответствии с которой он рассматривается как частично упорядоченное множество действий.

Отношение предшествования действий определяется как причинная зависимость действий в модели. Это отношение задает частичный порядок на действиях. Поэтому два действия параллельны, если они причинно независимы. Таким образом, параллельный процесс, элементы которого частично упорядочены отношением предшествования, может быть полностью представлен частично упорядоченным множеством. Поведение параллельных недетерминированных процессов описывается множеством их "чистых" параллельных подпроцессов. Каждый процесс в таком множестве- результат недетерминированного выбора между конфликтующими действиями. Но часто необходимо иметь дело с конфликтами семантического уровня и выражать поведение системы с конфликтами (или недетерминированного процесса) в качестве некоторого семантического объекта.

По этой причине вводится отношение альтернативы. Два действия a и b альтернативны,если выполнение a исключает выполнение b и наоборот.

Среди формальных моделей для описания параллельных систем и процессов алгебраические исчисления и логики процессов занимают особое место. В этих исчислениях процесс описывается алгебраической (или логической) формулой, и проверка свойств процесса выполняется посредством эквивалентностей,аксиом и правил вывода.

1. Синтаксис AFP_2

Пусть $\alpha=\{a,b,c,..\}$- конечный алфавит символов действий (базис действий процесса).

Действия комбинируются в процесс с помощью операций ; (предшествование), $\bar{\vee}$ (исключение или альтернатива) и $\|$ (параллельность).

В процессе (a;b) сначала выполняется действие a и только после этого- b.

Процесс $(a\bar{V}b)$ описывает два возможных поведения: если выбирается выполнение действия a, то b не случается и наоборот.

Формула $(a\|b)$ представляет процесс, в котором действия a и b выполняются параллельно.

Предполагается, что каждое действие имеет свое уникальное имя. Например, формула $(a;c)\|(b;c)$ описывает процесс, в котором действия a и b выполняются параллельно, а только затем выполняется c. Таким образом, выполнения действия c в подпроцессах (a;c) и (b;c) синхронизированны.

Алгебра AFP_2 содержит механизмы для описания как самих процессов, так и их свойств. Для выражения и проверки различных свойств процессов к базису действий процесса добавляются другие множества.

$\bar{\alpha}=\{\bar{a},\bar{b},\bar{c},...\}$ — двойной к α алфавит символов не-действий, которые выражают факты, что соответствующие действия не выполняются во время протекания процесса из-за выполнения некоторых альтернативных им действий.

$\Delta_\alpha=\{\delta_a,\delta_b,\delta_c,...\}$ — алфавит тупиковых действий, описывающий действия, которые не могут выполняться из-за некоторого противоречия или ошибки в описании процесса.

Вводятся также дополнительные операции: $\vee$ (дизъюнкция или объединение), $\urcorner$ ("не случится"), $\tilde{\urcorner}$ ("не случится ошибочно").

Формула $(P\vee Q)$ определяет процесс, в котором выполняется либо подпроцесс P, либо Q. В этом случае множество возможных поведений процесса-объединение множеств поведений подпроцессов P и Q.

Операция $\urcorner$ — модифицированное отрицание. $\urcorner P$ означает, что процесс P не случится, то есть выполняются не-действия из P.

Операция $\tilde{\urcorner}$ — другой тип отрицания. $\tilde{\urcorner}P$ означает, что процесс P не случается в результате некоторой ошибки, то есть любое действие из P не случается во время функционирования процесса в результате некоторых противоречивых требований в описании процесса.

Формула AFP_2 в базисе $\alpha\bar{\cup}\bar{\alpha}\cup\Delta_\alpha$ определяется так:

1) $a,\bar{a},\delta_a$, где $a\in\alpha$, $\bar{a}\in\bar{\alpha}$, $\delta_a\in\Delta_\alpha$ — элементарные формулы;

2) Если P и Q — формулы, тогда $(P\|Q)$, $(P\bar{V}Q)$, $(P\vee Q)$, $(P;Q)$, $\urcorner P$, $\tilde{\urcorner}P$ — формулы.

2. Денотационная семантика AFP_2

Семантика процесса, который описывается формулой AFP_2 есть множество частичных порядков, то есть совокупность частично упорядоченных множеств с порядком по предшествованию действий при выполнении данного процесса.

Частично упорядоченное множество — пара $(V,<)$, состоящая из:

-множества вершин, моделирующих действия, не-действия и тупиковые действия процесса, то есть $V\subseteq\alpha\bar{\cup}\bar{\alpha}\cup\Delta_\alpha$;

-частичного порядка над V, где $a<b$ интерпретируется так: действие a обязательно предшествует b в процессе.

Над частично упорядоченными множествами вводятся операции $;$, $\bar{V}$, $\|$, $\urcorner$, $\tilde{\urcorner}$, и операция модифицированного объединения $\tilde{\cup}$, определяемые в [Ch89],[Ch90-1].

Пусть $\mathbb{D}_2[P]$-совокупность частично упорядоченных множеств, связанных с процессом P. <u>Денотационная семантика</u> <u>AFP_2</u> определяется так:

1) $\mathbb{D}_2[a]=(\{a\},\emptyset)$, $\mathbb{D}_2[\bar{a}]=(\{\bar{a}\},\emptyset)$, $\mathbb{D}_2[\delta_a]=(\{\delta_a\},\emptyset)$

2) $\mathbb{D}_2[P \| Q]=\mathbb{D}_2[P] \| \mathbb{D}_2[Q]$

3) $\mathbb{D}_2[P;Q]=\mathbb{D}_2[P];\mathbb{D}_2[Q]$

4) $\mathbb{D}_2[P \bar{\vee} Q]=\mathbb{D}_2[P] \bar{\vee} \mathbb{D}_2[Q]$

5) $\mathbb{D}_2[P \vee Q]=\mathbb{D}_2[P] \tilde{\vee} \mathbb{D}_2[Q]$

6) $\mathbb{D}_2[\underset{\sim}{\neg} P]=\underset{\sim}{\neg}\mathbb{D}_2[P]$

7) $\mathbb{D}_2[\neg P]=\neg\mathbb{D}_2[P]$

Пример: Рассмотрим процесс, описываемый формулой $P=(a\bar{\vee}b) \| (b\bar{\vee}c)$. Тогда $\mathbb{D}_2[P]=\{(\{b,\bar{a},\bar{c}\},\emptyset),(\{a,c,\bar{b}\},\emptyset)\}$ состоит из двух частично упорядоченных множеств.

3. <u>Аксиоматизация</u> <u>AFP_2</u>

Рассмотренная семантика для процессов задает эквивалентность. Два процесса P и Q <u>эквивалентны</u>, $P\approx_e Q$, тогда и только тогда, когда $\mathbb{D}_2[P]=\mathbb{D}_2[Q]$.

В [Ch90-1,c.4,p.14] вводится понятие <u>контекста</u> $\mathbb{C}[]$. Это выражение с нулем или более пустых "дыр", которые могут быть заполнены другими выражениями. $\mathbb{C}[P]$ представляет результат помещения формулы P в каждую "дыру".

Два процесса, описываемые формулами P и Q, <u>конгруэнтны</u>, $P\approx_{co}Q$, тогда и только тогда, когда $\mathbb{C}[P]\approx_e\mathbb{C}[Q]$ для любого контекста $\mathbb{C}[]$.

В [Ch90-1,c.5,p.17] доказывается следующая лемма.

Лемма: $P\approx_e Q \Leftrightarrow P\approx_{co} Q$

Вводится <u>система</u> <u>аксиом</u> θ_2 в соответствии с отношением эквивалентности $\approx_e$.

В следующих равенствах P, Q, R обозначают формулы AFP_2, $a\in\alpha$, $\bar{a}\in\bar{\alpha}$, $\delta_a\in\Delta_\alpha$.

1.Ассоциативность

1.1 $P\|(Q\|R)=(P\|Q)\|R$

1.2 $P\bar{\vee}(Q\bar{\vee}R)=(P\bar{\vee}Q)\bar{\vee}R$

1.3 $P\vee(Q\vee R)=(P\vee Q)\vee R$

1.4 $P;(Q;R)=(P;Q);R$

2.Коммутативность

2.1 $P\|Q=Q\|P$

2.2 $P\bar{\vee}Q=Q\bar{\vee}P$

2.3 $P\vee Q=Q\vee P$

3.Дистрибутивность

3.1 $(P \parallel Q);R=(P;R)\parallel(Q;R)$

3.2 $P;(Q\parallel R)=(P;Q)\parallel(P;R)$

3.3 $(P \vee Q);R=(P;R)\vee(Q;R)$

3.4 $P;(Q\vee R)=(P;Q)\vee(P;R)$

3.5 $(P\vee Q)\parallel R=(P\parallel R)\vee(Q\parallel R)$

3.6 $P\bar{V}(Q\parallel R)=(P\bar{V}Q)\parallel(P\bar{V}R)$

4.Аксиомы для $\bar{V}$ и $\neg$

4.1 $P\bar{V}Q=(P\parallel \neg Q)\vee(\neg P\parallel Q)$

4.2 $\neg(P\parallel Q)=(\neg P)\parallel(\neg Q)$

4.3 $\neg(P\vee Q)=(\neg P)\vee(\neg Q)$

4.4 $\neg(P;Q)=(\neg P)\parallel(\neg Q)$

4.5 $\neg a=\bar{a}$

4.6 $\neg\bar{a}=\bar{a}$

4.7 $\neg\delta_a=\bar{a}$

5.Структурные свойства

5.1 $\bar{a};P=\bar{a}\parallel P$

5.2 $P;\bar{a}=P\parallel\bar{a}$

5.3 $P\parallel(P;Q)=(P;Q)$

5.4 $Q\parallel(P;Q)=(P;Q)$

5.5 $P;Q;R=(P;Q)\parallel(Q;R)$

5.6 $(P;Q)\parallel(Q;R)=(P;Q)\parallel(Q;R)\parallel(P;R)$

5.7 $P\parallel P=P$

5.8 $P\vee P=P$

5.9 $\tilde{\neg}P\vee P=P$

6.Аксиомы для тупиковых действий и $\tilde{\neg}$

6.1 $a\parallel\bar{a}=\delta_a$

6.2 $a;a=\delta_a$

6.3 $a\parallel\delta_a=\delta_a$

6.4 $\delta_a;P=\delta_a\parallel\tilde{\neg}P$

6.5 $P;\delta_a=P\parallel\delta_a$

6.6 $\delta_a\parallel\neg P=\delta_a\parallel\tilde{\neg}P$

6.7 $\tilde{\neg}a=\delta_a$

6.8 $\tilde{\neg}\bar{a}=\delta_a$

6.9 $\tilde{\neg}\delta_a=\delta_a$

6.10 $\tilde{\neg}(P\parallel Q)=\tilde{\neg}P\parallel\tilde{\neg}Q$

6.11 $\tilde{\neg}(P;Q)=\tilde{\neg}P\parallel\tilde{\neg}Q$

6.12 $\tilde{\neg}(P\vee Q)=\tilde{\neg}P\vee\tilde{\neg}Q$

Для доказательства полноты системы аксиом θ_2 вводится понятие канонической формы формулы AFP_2.

4. Каноническая форма формулы AFP_2

Определим подмножество $\alpha(P)$ символов действий из α, встречающихся в формуле P :

$\alpha(a)=a$

$\alpha(\bar{a})=a$

$\alpha(\delta_a)=a$

$\alpha(P \circ Q)=\alpha(P)\cup\alpha(Q), \circ\in\{;, \|, \overline{\vee}, {}_{\vee}\}$

$\alpha(\neg_{\|}P)=\alpha(P)$

$\alpha(\tilde{\neg}_{\|}P)=\alpha(P)$

В дополнение к понятию $\alpha(P)$, введенному Л. А. Черкасовой [Ch90-1, c.6, p.19], я определяю следующее важное понятие содержимого формулы.

Содержимое формулы P, cont(P), - множество символов из $\alpha(P)\cup\overline{\alpha}(P)\cup\Delta_{\alpha}(P)$, определяемое так:

1) $cont(a)=\{a\}, cont(\bar{a})=\{\bar{a}\}, cont(\delta_a)=\{\delta_a\}$;

2) $cont(P \circ Q)=cont(P)\cup cont(Q)$, где $\circ\in\{;, \|, \overline{\vee}, {}_{\vee}\}$, P и Q не содержат символов $\neg_{\|}$ и $\tilde{\neg}_{\|}$.

Пусть $\overline{\alpha}(P)$ - алфавит, двойной к $\alpha(P)$: $\overline{\alpha}(P)=\{\bar{a} | a\in\alpha(P)\}$ и $\Delta(P)=\{\delta_a | a\in\alpha(P)\}$

;-предшествование - формула вида $P_1;..;P_n = \mathop{;}_{i=1}^{n} P_i, P_i \in\alpha\cup\overline{\alpha}\cup\Delta_{\alpha} (1\le i\le n)$.

$\|$-конъюнктивный терм - формула, содержащая только операции $\|$ и ; над символами объединенного алфавита $\alpha\cup\overline{\alpha}\cup\Delta_{\alpha}$.

$\|$-конъюнкция - $\|$-конъюнктивный терм, имеющий вид $P_1\|..\|P_n = \mathop{\|}_{i=1}^{n} P_i$.

Нормальная $\|$-конъюнкция - $\|$-конъюнкция, для которой истинны следующие утверждения:

1) каждая формула P_i $(1\le i\le n)$ имеет одну из следующих форм:

- элементарная формула a $(a\in\alpha)$, $\bar{a}$ $(a\in\alpha)$, δ_a $(\delta_a\in\Delta_{\alpha})$;
- элементарное предшествование (a;b), где $a,b\in\alpha$ и $a\ne b$;

2) если имеется формула P_i $(1\le i\le n)$ в форме δ_a $(\delta_a\in\Delta_{\alpha})$, тогда нет другой формулы P_j $(1\le j\le n)$ такой, что $P_j=\bar{b}$ $(b\in\alpha)$;

3) для любых формул P_i и P_j $(1\le i\ne j\le n)$ таких, что $\alpha(P_i)\cap\alpha(P_j)\ne\emptyset$ P_i и P_j должны иметь форму различных элементарных предшествований;

4) для любой пары $P_i=(a;b)$ и $P_j=(b;c)$ $(1\le i\ne j\le n)$ существует терм $P_k=(a;c)$, описывающий транзитивное замыкание отношения

предшествования для действий a,b и c.

Назовем 1 (или 2 или 3 или 4)-$\|$-<u>конъюнкцией</u> $\|$-конъюнкцию, удовлетворяющую соответственно условиям 1(или 2 или 3 или 4) из определения нормальной $\|$-конъюнкции. Аналогично введем определение, например, 1,2,3-$\|$-<u>конъюнкции</u>. В соответствии с этими определениями нормальная $\|$-конъюнкция есть 1,2,3,4-$\|$-конъюнкция.

Формула P имеет <u>каноническую</u> <u>форму</u>, если $P=P_1\vee\dots\vee P_n=\bigvee_{i=1}^{n} P_i$, то есть P-$_\vee$-<u>дизъюнкция</u>, где:

1)P_i $(1\le i\le n)$-нормальная $\|$-конъюнкция;

2)любые P_i и P_j $(1\le i\ne j\le n)$ различны;

3)любые P_i и P_j $(1\le i\ne j\le n)$ <u>не</u> <u>префиксны</u> друг другу, то есть если

$\alpha(P_i)\supseteq\alpha(P_j), cont(P_i)\subseteq\alpha\cup\bar{\alpha}, cont(P_j)\subseteq\alpha\cup A_{\alpha}, cont(P_i)\cap\alpha\supseteq cont(P_j)\cap\alpha$,

в формуле не должно быть дизъюнктивного члена P_j.

Аналогично определениям для $\|$-конъюнкций вводим определения 1 (или 2 или 3 или 4)-$_\vee$-<u>дизъюнкций</u> и другие. Таким образом, каноническая форма- 1,2,3-$_\vee$-дизъюнкция. В дальнейшем будем понимать под дизъюнкцией $_\vee$-дизъюнкцию, под коньюнкцией- $\|$-конъюнкцию, а под предшествованием- ;-предшествование.

Я счел необходимым ввести также следующее понятие.

<u>Конъюнкция</u> (<u>дизъюнкция</u>) <u>максимальна</u>, если она не является конъюнктивным (дизъюнктивным) членом никакой другой конъюнкции (дизъюнкции).

Пример: В формуле $(((a;b)\|c)\vee d)\|e$ $(a;b)$ и c - не максимальные конъюнкции, так как они- конъюнктивные члены максимальной конъюнкции $((a;b)\|c)$. Вся формула- не конъюнкция, потому что ее левый конъюнктивный член- дизъюнкция.

Заметим, что конъюнкция характеризует одно из возможных поведений альтернативного процесса и является представлением частичного порядка этого процесса.

Пример: Формула $(a\|\bar{b})\vee(\bar{a}\|b)$ находится в канонической форме.

Запись $A=_{\theta_2}B$ означает, что равенство формул A и B алгебры AFP_2 может быть доказано с использованием системы аксиом θ_2.

Канонические формы A и B <u>изоморфны</u> тогда и только тогда, когда A может быть сведена к B (и наоборот) с использованием аксиом коммутативности и ассоциативности для операций $\|$ и $\vee$.

Пример: Формулы $(a\|b\|\bar{c})\vee(c\|\bar{a}\|\bar{b})$ и $(\bar{a}\|\bar{b}\|c)\vee(b\|a\|\bar{c})$ изоморфны.

Теорема1[Ch90-1,c.6,p.21]: Любая формула AFP_2 может быть сведена к единственной до изоморфизма канонической форме.

Главный результат сформулирован в виде следующей теоремы [Ch90-1,c.6,p.22].

Теорема2: Для любых формул P и Q алгебры AFP_2 истинно

$$P\approx_e Q \Leftrightarrow P=_{\theta_2}Q$$

Таким образом, для любых двух формул P и Q алгебры AFP_2 мы можем выяснить, эквивалентны ли они, то есть описываются ли они одной и той же совокупностью частично упорядоченных множеств. Для этого достаточно свести формулы P и Q к их каноническим формам P' и Q' и проверить их на изоморфизм.

5. <u>Система</u> <u>правил</u> <u>переписывания</u> <u>RWS_2</u>

Процесс приведения формулы AFP_2 к каноническому виду с помощью аксиом системы θ_2 иногда становится трудоемким и нетривиальным из-за того, что эквивалентности приходится применять в ту и другую сторону.

Хотелось бы иметь систему направленных правил, которые приводили бы формулу к нужному виду. Процесс приведения желательно автоматизировать. Для этого нужно создать систему переписывания без циклов (то есть процесс приведения должен завершаться за конечное время), приводящую исходную формулу в один из изоморфных между собой канонических видов.

В соответствии с этими требованиями мною построена <u>система правил переписывания</u> RWS_2. Перед описанием этой системы введем необходимое определение.

<u>Замена</u> <u>в</u> <u>формуле</u> P <u>подформулы</u> P_i <u>на</u> Q, $[P]_Q^{P_j}$, есть формула $P_1 \circ .. P_{i-1} \circ Q \circ P_{i+1} \circ .. \circ P_n$, где $P = P_1 \circ .. \circ P_{i-1} \circ P_i \circ P_{i+1} \circ .. \circ P_n$, $\circ \in \{;, \|, \bar{\vee}, {}_{\vee}\}$.

В следующих правилах RWS_2 P, Q, R обозначают формулы AFP_2, $a, b, c \in \alpha$, $\bar{a}, \bar{b} \in \bar{\alpha}$, $\delta_{\bar{a}}, \delta_b \in \Delta_{\bar{\alpha}}$, а цифры в скобках - номера равенств системы θ_2, которые использовались при построении соответствующих правил.

1.1 $\circ \in \{;, \|, {}_{\vee}\} \Rightarrow$

$P \circ (Q \circ R) \to (P \circ Q) \circ R$

(1.1, 1.3, 1.4)

2.1 $(\bullet, \circ) \in \{(\|, ;), ({}_{\vee}, ;), ({}_{\vee}, \|)\} \Rightarrow$

$(P \circ Q) \bullet R \to (P \bullet R) \circ (Q \bullet R)$

(3.1, 3.3, 3.5)

2.2 $(\bullet, \circ) \in \{(\|, ;), ({}_{\vee}, ;), ({}_{\vee}, \|)\} \Rightarrow$

$P \bullet (Q \circ R) \to (P \bullet Q) \circ (P \bullet R)$

(2.1, 3.2, 3.4, 3.5)

3.1 $P \bar{\vee} Q \to (P \| (\neg_{\|} Q)) {}_{\vee} ((\neg_{\|} P) \| Q)$

(4.1)

4.1 $\circ \in \{\|, ;\}, \neg \in \{\neg_{\|}, \tilde{\neg}_{\|}\} \Rightarrow$

$\neg(P \circ Q) \to (\neg P) \| (\neg Q)$

(4.2, 4.4, 6.10, 6.11)

4.2 $\neg \in \{\neg_{\|}, \tilde{\neg}_{\|}\} \Rightarrow$

$\neg(P {}_{\vee} Q) \to (\neg P) {}_{\vee} (\neg Q)$

(4.3,6.12)

4.3 $P=a$ или $P=\bar{a}$ или $P=\delta_a \Rightarrow$

$\rceil P \to \bar{a}$

(4.5,4.6,4.7)

4.4 $P=a$ или $P=\bar{a}$ или $P=\delta_a \Rightarrow$

$\tilde{\rceil} P \to \delta_a$

(6.7,6.8,6.9)

5.1 $P,Q,R \in \alpha \cup \bar{\alpha} \cup \Delta_\alpha \Rightarrow$

$(P;Q);R \to ((P;Q) \| (Q;R)) \| (P;R)$

(5.5,5.6)

5.2 $Q \in \alpha \cup \bar{\alpha} \cup \Delta_\alpha \Rightarrow$

$\bar{a};Q \to \bar{a} \| Q$

(5.1)

5.3 $P \in \alpha \cup \bar{\alpha} \cup \Delta_\alpha \Rightarrow$

$P;\bar{a} \to P \| \bar{a}$

(5.2)

5.4 $a;a \to \delta_a$

(6.2)

5.5 $Q=b$ или $Q=\bar{b}$ или $Q=\delta_b \Rightarrow$

$\delta_a;Q \to \delta_a \| \delta_b$

(6.4,6.7,6.8,6.9)

5.6 $P \in \alpha \cup \bar{\alpha} \cup \Delta_\alpha \Rightarrow$

$P;\delta_a \to P \| \delta_a$

(6.5)

6.1 P– 1-конъюнкция, $(a;b)=P'$– конъюнктивный член P, в максимальной 1-конъюнкции, содержащей P в качестве конъюнктивного члена, нет члена $(a;c)=P'' \Rightarrow$

$P \| (b;c) \to (P \| (b;c)) \| (a;c)$

(2.1,5.6)

6.2 P– 1-конъюнкция,$(c;a)=P'$– конъюнктивный член P, в максимальной 1-конъюнкции, содержащей P в качестве конъюнктивного члена, нет члена $(b;a)=P'' \Rightarrow$

$P \| (b;c) \to (P \| (b;c)) \| (b;a)$

(2.1,5.6)

7.1 P– 1,4-конъюнкция,P'–конъюнктивный член P,$P'=a$ или $P'=b \Rightarrow$

$P \| (a;b) \to [P]^{P'}_{(a;b)}$

(2.1,5.3,5.4)

7.2 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=(a;b) или P'=(b;a) ⇒

$P\|a\to P$

(2.1,5.3,5.4)

7.3 P– 1,4-конъюнкция,P'-конъюнктивный член P, P'=a или P'=δ_a ⇒

$P\|\bar{a}\to[P]^{P'}_{\delta_a}$

(2.1,6.1)

7.4 P– 1,4-конъюнкция,P'=$\bar{a}$ или P'=δ_a ⇒

$P\|a\to[P]^{P'}_{\delta_a}$

(2.1,6.1,6.3)

7.5 P– 1,4-конъюнкция,P'-конъюнктивный член P, P'=a или P'=$\bar{a}$ ⇒

$P\|\delta_a\to[P]^{P'}_{\delta_a}$

(2.1,6.3)

7.6 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=(a;b) ⇒

$P\|\bar{a}\to[P]^{P'}_{\delta_b}\|\delta_a$

(1.1,1.4,2.1,3.1,5.1,5.2,5.7,6.1,6.4,6.7,6.11)

7.7 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=(b;a) ⇒

$P\|\bar{a}\to[P]^{P'}_{b}\|\delta_a$

(1.4,2.1,5.2,6.1,6.5)

7.8 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=(a;b) ⇒

$P\|\delta_a\to[P]^{P'}_{\delta_b}\|\delta_a$

(1.1,1.4,2.1,5.4,5.7,6.4,6.7,6.11)

7.9 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=(b;a) ⇒

$P\|\delta_a\to[P]^{P'}_{b}\|\delta_a$

(1.4,2.1,6.5)

7.10 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=$\bar{a}$ или P'=δ_a ⇒

$P\|(a;b)\to[P]^{P'}_{\delta_a}\|\delta_b$

(1.4,2.1,5.1,5.2,5.7,6.1,6.4,6.7)

7.11 P– 1,4-конъюнкция,P'-конъюнктивный член P,P'=$\bar{a}$ или P'=δ_a ⇒

$P\|(b;a)\to[P]^{P'}_{\delta_a}\|b$

(1.1,1.4,2.1,3.1,5.1,5.7,6.1,6.5,6.7,6.11)

7.12 P– 1,4-конъюнкция,P'=Q-конъюнктивный член P ⇒

$P \| Q \to P$

(5.7)

8.1 P– 1,3,4-конъюнкция, $P' = \delta_a$ –конъюнктивный член P $\Rightarrow$

$P \| \bar{b} \to P \| \delta_b$

(2.1,4.5,6.6,6.8)

8.2 P– 1,3,4-конъюнкция, $P' = \bar{b}$–конъюнктивный член P $\Rightarrow$

$P \| \delta_a \to [P]^{P'}_{\delta_b} \| \delta_a$

(2.1,4.5,6.6,6.8)

9.1 P– 1-дизъюнкция, $P' = Q$–дизъюнктивный член P $\Rightarrow$

$P \vee Q \to P$

(2.3,5.8)

10.1 P– 1,2-дизъюнкция, Q–нормальная конъюнкция, P'–дизъюнктивный член P, $\alpha(P') \geq \alpha(Q)$, $cont(P') \leq \alpha \cup \bar{\alpha}$, $cont(Q) \leq \alpha \cup \Delta_{\alpha}$,

$cont(P') \cap \alpha \geq cont(Q) \cap \alpha \Rightarrow$

$P \vee Q \to P$

(2.3,5.9)

10.2 P– 1,2-дизъюнкция, Q–нормальная конъюнкция, P'–дизъюнктивный член P, $\alpha(P') = \alpha(Q)$, $cont(Q) \leq \alpha \cup \bar{\alpha}$, $cont(P') \leq \alpha \cup \Delta_{\alpha}$,

$cont(P') \cap \alpha \leq cont(Q) \cap \alpha \Rightarrow$

$P \vee Q \to [P]^{P'}_{Q}$

(2.3,5.9)

Сделаем краткий обзор системы правил переписывания.

Для избежания бесконечных цепочек вывода вида $P \circ (Q \circ R) \to (P \circ Q) \circ R \to P \circ (Q \circ R) \to \ldots$, $\circ \in \{;, \|, \vee\}$ вводится правило левой ассоциативности 1.1.

В систему RWS_2 нельзя включать правила коммутативности, применение которых может привести к бесконечным цепочкам вида $P \circ Q \to Q \circ P \to P \circ Q \to \ldots$, $\circ \in \{\|, \vee\}$. Поэтому дополнительно вводятся симметричные правила, необходимые при отсутствии правил коммутативности. На этой идее основаны правила дистрибутивности группы 2.

Пример: В системе θ_2 нет аксиомы, симметричной аксиоме 3.5 $(P \vee Q) \| R = (P \| R) \vee (Q \| R)$. Поэтому, если мы не включим в нашу систему правил нового симметричного правила, основанного на аксиоме $P \| (Q \vee R) = (P \| Q) \vee (P \| R)$, мы не сможем преобразовать формулу $a \| (b \vee c)$ к виду $(a \| b) \vee (a \| c)$.

Правило 3.1 позволяет избавиться от символа $\bar{\vee}$, а правила группы 4 –от символов Π и $\tilde{\Pi}$.

Правила группы 5 используются для удовлетворения свойству 1 нормальной конъюнкции.

В связи с отсутствием правил коммутативности возникают и трудности другого рода, связанные с удалением друг от друга конъюнктивных (дизъюнктивных) членов формулы, к которым можно применить некоторую аксиому. Тогда, в дополнение к подходу, основанному на введении симметричных правил, применяется другой. Рассмотрим формулу $P \parallel Q$ (или $P_{\vee}Q$). В подформуле P, являющейся конъюнкцией (дизъюнкцией), ищется подформула P' такая, что к $P' \parallel Q$ (к $P'_{\vee}Q$) применима некоторая аксиома из θ_2.

На этой идее основаны правила группы 6, предназначенные для удовлетворения свойству 4 нормальной конъюнкции, группы 7, необходимые для выполнения свойства 3, правила группы 8 для удовлетворения свойству 2, а также правила групп 9 и 10, нужные для выполнения соответственно свойств 2 и 3 канонической формы.

Пример: Аксиома 6.1 ($a \parallel \bar{a} = \delta_a$) не применима при отсутствии правил коммутативности к формулам $a \parallel b \parallel \bar{a}$ и $\bar{a} \parallel b \parallel a$. Вводятся симметричные правила 7.3 и 7.4.

В первом случае, при применении правила 7.3 ($P=a \parallel b, Q=\bar{a}, P'=a$) получаем формулу $\delta_a \parallel b$.

Применяя правило 7.4 к второй формуле ($P=\bar{a} \parallel b, Q=a, P'=\bar{a}$), также получаем $\delta_a \parallel b$.

Кроме того, для избежания бесконечных цепочек вывода вида $(a;b) \parallel (b;c) \to ((a;b) \parallel (b;c)) \parallel (a;c) \to (((a;b) \parallel (b;c)) \parallel (a;c)) \parallel (a;c) \to \dots$, в правилах группы 6 конъюнктивный член-транзитивное замыкание отношения предшествования ищется в максимальной 1-конъюнкции, содержащей данную. Правило применяется только тогда, когда такого члена в максимальной 1-конъюнкции нет. Этот прием предохраняет от бесконечного увеличения длины формулы в процессе ее приведения.

Необходимо отметить, что правила 7.6-7.11, 8.1, 8.2 основаны на новых аксиомах, полученных из аксиом θ_2:

$\bar{a} \parallel (a;b) = \delta_a \parallel \delta_b$

$\delta_a \parallel (a;b) = \delta_a \parallel \delta_b$

$\bar{a} \parallel (b;a) = \delta_a \parallel b$

$\delta_a \parallel (b;a) = \delta_a \parallel b$

$\delta_a \parallel \bar{b} = \delta_a \parallel \delta_b$

Пример: Первая из аксиом получается по следующей цепочке равенств, в которой над знаками равенств указаны номера применяемых аксиом, а звездочка обозначает их применение справа налево:

$$\bar{a} \parallel (a;b) \overset{5.1*}{=\!=} \bar{a};(a;b) \overset{1.4}{=\!=} (\bar{a};a);b \overset{5.1}{=\!=} (\bar{a} \parallel a);b \overset{2.1}{=\!=} (a \parallel \bar{a});b \overset{6.1}{=\!=} \delta_a;b \overset{6.4}{=\!=}$$

$$\delta_a \parallel (\tilde{\ }_{\parallel} b) \overset{6.7}{=\!=} \delta_a \parallel \delta_b.$$

Рассмотрим пример на свойство 3 канонической формы.

Пример: Формула

$$(a \parallel c \parallel \bar{b} \parallel \bar{d} \parallel \bar{e})_{\vee}(c \parallel \delta_b \parallel \delta_d \parallel \delta_a \parallel \delta_e)_{\vee}(a \parallel \delta_b \parallel \delta_e \parallel \delta_d \parallel \delta_c)_{\vee}((b;d) \parallel (b;e) \parallel \bar{a} \parallel \bar{c})$$

приводится с помощью правил группы 10 к виду, удовлетворяющему свойству 3 канонической формы: $(a\|c\|\bar{b}\|\bar{d}\|\bar{e})\bar{\vee}((b;d)\|(b;e)\|\bar{a}\|\bar{c})$.

Докажем серию важных утверждений о системе RWS_2.

Утверждение1: Если к формуле AFP_2 не применимо ни одно из правил групп 1-5, то она является дизъюнкцией 1-конъюнкций.

Доказательство:

Заметим, что, если к формуле не применимо правило 1.1, то все скобки, объединяющие подформулы, соединенные одинаковыми знаками операций, смещены влево. В дальнейшем будем предполагать, что все формулы уже обладают этим свойством.

Если к формуле не применимы правила 1.1 и 3.1, то в этой формуле нет операции $\bar{\vee}$.

Очевидно также, что с помощью правил группы 4 мы избавляемся от Π и $\tilde{\Pi}$. Если в формуле есть эти символы, стоящие перед сложными подформулами, то к ней обязательно применимы правила группы 4, смещающие знаки операций Π и $\tilde{\Pi}$ к элементарным подформулам видов $a, \bar{a}, \delta_a$. Затем, при применении правил 4.3 и 4.4, знаки Π и $\tilde{\Pi}$ исчезают. Итак, если к формуле не применимы правила групп 1,3,4, эта формула-без символов $\bar{\vee}$, Π и $\tilde{\Pi}$.

Если, в дополнение к этому, к формуле не применимы правила группы 2, приводящие формулу к виду $P=\bigvee_{i=1}^{m}\|_{j=1}^{n_i} P_{ij}$, то формула является дизъюнкцией конъюнкций с членами P_{ij}, являющимися предшествованиями или элементарными формулами.

Если, к тому же, к формуле не применимо правило 5.1, то можно заключить, что она-дизъюнкция конъюнкций с членами вида $P_{ij}=(Q_{ij};R_{ij})$ или $P_{ij}=Q_{ij}$, где $Q_{ij}, R_{ij} \in \alpha\cup\bar{\alpha}\cup\Delta_\alpha$.

Правила 5.2-5.6 позволяют избавиться от символов из $\bar{\alpha}\cup\Delta_\alpha$ и одинаковых символов в двучленных предшествованиях. Таким образом, если к формуле не применимы правила групп 1-5, она является дизъюнкцией конъюнкций с членами вида $a, \bar{a}, \delta_a$ или $(a;b), a\neq b$, то есть эта формула-дизъюнкция 1-конъюнкций.

■

Утверждение2: Если к формуле AFP_2 не применимо ни одно из правил групп 1-6, то она является дизъюнкцией 1,4-конъюнкций.

Доказательство:

По утверждению 1, наша формула-дизъюнкция 1-конъюнкций. Так как к этой формуле не применимы правила группы 6, дополняющие ее 1-конъюнкции новыми членами-транзитивными замыканиями отношения предшествования, то, по определению, дизъюнктивные члены этой формулы-1,4-конъюнкции.

■

Утверждение3: Если к формуле AFP_2 не применимо ни одно из правил

групп 1-7, она является дизъюнкцией 1,3,4-конъюнкций.
Доказательство:

По утверждению 2, формула- дизъюнкция 1,4-конъюнкций. Достаточно доказать, что из неприменимости правил группы 7 следует выполнение условия 3 нормальной конъюнкции для всех дизъюнктивных членов формулы. Рассмотрим ситуации, когда для дизъюнктивного члена формулы, 1,4-конъюнкции $P_i = \Pi_{j=1}^{n} P_{ij}$, выполняется условие: $\alpha(P_{ik})\cap\alpha(P_{il})\neq\emptyset$ $(1\leq k\neq l\leq n)$ и P_{ik}, P_{il} не являются различными элементарными предшествованиями.

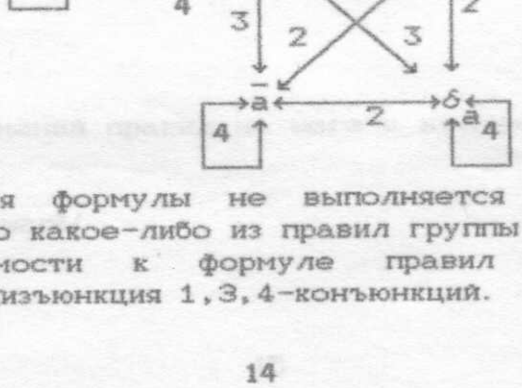
1) Конъюнктивные члены имеют вид a и (a;b) или b и (a;b). Тогда к конъюнкции, содержащей эти члены, применимо правило 7.1 или 7.2, после применения которого $\alpha(P_{ik})\cap\alpha(P_{il})=\emptyset$.

2) Конъюнктивные члены имеют вид a и $\bar{a}$ или a и δ_a или $\bar{a}$ и δ_a. Тогда к соответствующей конъюнкции применимо одно из правил 7.3, 7.4, 7.5. После его применения $\alpha(P_{ik})\cap\alpha(P_{il})=\emptyset$.

3) Конъюнктивные члены имеют вид $\bar{a}$ и (a;b) или b и (a;b) или δ_a и (a;b) или δ_b и (a;b). Тогда к конъюнкции применимо одно из правил 7.6-7.11, преобразующих конъюнкцию к виду, где $\alpha(P_{ik})\cap\alpha(P_{il})=\emptyset$.

4) Конъюнктивные члены имеют вид a и a или $\bar{a}$ и $\bar{a}$ или δ_a и δ_a или (a;b) и (a;b), то есть совпадают. Тогда к конъюнкции применимо правило 7.12, уничтожающее лишний конъюнктивный член и выполняющее условие $\alpha(P_{ik})\cap\alpha(P_{il})=\emptyset$.

Связи между конъюнктивными членами наглядно представляются следующей схемой, в которой стрелки с номерами пунктов доказательства этого утверждения соединяют формулы, которые могут быть на месте P_{ik} и P_{il}. Эта схема показывает, что в утверждении рассмотрены все случаи невыполнимости условия 3 нормальной конъюнкции.



Значит, если для формулы не выполняется условие 3, к ней обязательно применимо какое-либо из правил группы 7.
Итак, при неприменимости к формуле правил групп 1-7 можно утверждать, что она-дизъюнкция 1,3,4-конъюнкций.

Утверждение4: Если к формуле AFP_2 не применимо ни одно из правил групп 1-8, то она является 1-дизъюнкцией.
Доказательство:

По утверждению 3, формула- дизъюнкция 1,3,4-конъюнкций. Покажем, что из неприменимости правил группы 8 следует выполнение условия 2 нормальной конъюнкции для всех ее дизъюнктивных членов. Это очевидно, так как правило 8.1 или 8.2 применимо к какому-то дизъюнктивному члену P_i формулы P, если в нем есть конъюнктивные члены P_{ik} и P_{il}, имеющие вид δ_a и $\bar{b}$ (a≠b по свойству 3 нормальной конъюнкции). В заключение заметим, что 1,2,3,4-конъюнкция-нормальная конъюнкция, а дизъюнкция нормальных конъюнкций-1-дизъюнкция.

■

Утверждение5: Если к формуле AFP_2 не применимо ни одно из правил групп 1-9, то она-1,2-дизъюнкция.
Доказательство:

По утверждению 4, формула-1-дизъюнкция. Правило 9.1 не применимо, если все дизъюнктивные члены различны, то есть если формула-1,2-дизъюнкция.

■

Утверждение6: Если к формуле AFP_2 не применимо ни одно из правил RWS_2, она находится в канонической форме.
Доказательство:

По утверждению 5, формула-1,2-дизъюнкция. Очевидно, что правила группы 10 не применимы только тогда, когда формула-не 1,2,3-дизъюнкция, то есть не находится в канонической форме.

■

6.Описание программы CANONIC

Программа CANONIC, занимающая около 1000 строк на языке Си, предназначена для преобразования формулы AFP_2 в дизъюнкцию 1-конъюнкций и основана на утверждении 1 главы 5.

Тело функции main имеет вид:

```
вывод предварительной информации о предназначении программы
   и формате вводимой формулы;
представление формулы в виде цепочки;
выдача сообщения "формула считана";
преобразование цепочки в дерево;
уничтожение цепочки;
печать формулы;
step=1;/*номер шага*/
do
{
 вывод step;
 nar=0;/*число применений правил на шаге с номером step*/
 применение правил;
 вывод nar;
 step++;/*следующий шаг*/
}
while(nar!=0);
```

вывод формулы,полученной в результате выполнения правил;

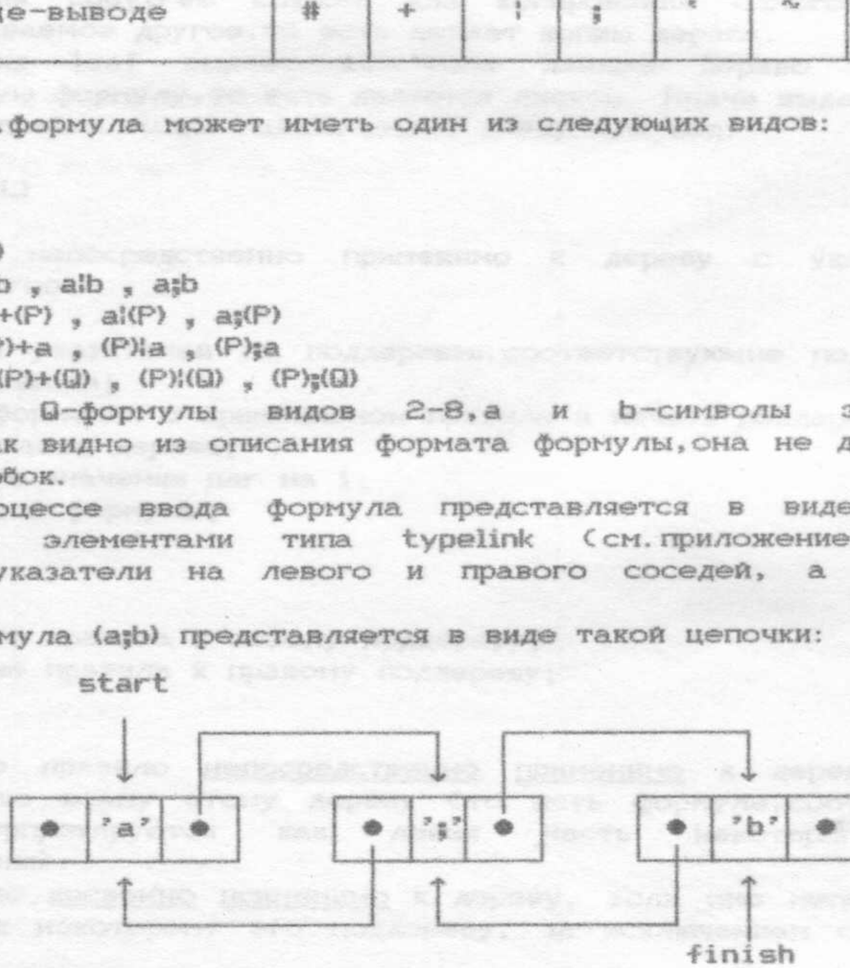
Формула должна вводиться по определенному формату. Символы $\bar{\vee}, \vee, \|, ;, \urcorner, \tilde{\urcorner}, \bar{\ }, \delta$ заменяются на имеющиеся на клавиатуре в соответствии с таблицей:

Исходное обозначение символа	$\bar{\vee}$	$\vee$	$\|$	;	$\urcorner$	$\tilde{\urcorner}$	$\bar{\ }$	δ
Имя символьной константы	ALT	DSJ	CNC	PRC	NOC	MNO	NOT	DLT
Изображение символа при вводе-выводе	#	+	!	;	'	~	–	*

Кроме того,формула может иметь один из следующих видов:
1) a
2) –a , *a
3) 'a , ~a
4) '(P) , ~(P)
5) a#b , a+b , a!b , a;b
6) a#(P) , a+(P) , a!(P) , a;(P)
7) (P)#a , (P)+a , (P)!a , (P);a
8) (P)#(Q) , (P)+(Q) , (P)!(Q) , (P);(Q)
где P и Q-формулы видов 2-8,a и b-символы элементарных действий. Как видно из описания формата формулы,она не должна иметь внешних скобок.

В процессе ввода формула представляется в виде двусвязной цепочки с элементами типа typelink (см.приложение), которые содержат указатели на левого и правого соседей, а также один символ.

Пример: Формула (a;b) представляется в виде такой цепочки:



Затем,с помощью функции transfct,по цепочке строится представление формулы в виде дерева с вершинами типа typevert,содержащими указатели на отца,1 и 2 сыновей,а также два символа,которые могут представлять элементарную формулу,то есть формулу видов 1 и 2. Таким образом,формула преобразуется в дерево с указателем на корень ancestor, с промежуточными вершинами,которые соответствуют функциональным символам,и листьями,представляющими элементарные формулы. В этом представлении подформулам соответствуют поддеревья.

После этого,для экономии памяти,цепочка уничтожается функцией dispchain и формула,представленная в виде дерева,печатается с помощью функции writetree (проверка правильности преобразования в дерево).

Потом начинается процесс применения к формуле правил групп 1-5. На каждом шаге с номером step просматриваются все правила, и, если они применимы, применяются, а число прменений правил на данном шаге, nar, увеличивается. В конце шага проверяется, равно ли число примененных на нем правил нулю. Если это так, то ни одно из правил групп 1-5 не применимо к формуле, а это, по утверждению 1 главы 5 означает, что она является дизъюнкцией 1-конъюнкций. В этом случае циклическое применение правил заканчивается, формула выводится на экран функцией writetree, и программа заканчивает работу. В противном случае номер шага увеличивается на единицу, и к формуле опять применяются правила.

Рассмотрим остальные функции, используемые в программе.

Функция copytree служит для копирования одного дерева во вновь создаваемое другое, то есть делает копию дерева.

Функция leaf выдает 1, если данное дерево представляет элементарную формулу, то есть является листом. Иначе выдается 0.

Тела правил rule11-rule56 имеют следующий вид.

```
if(root!=NULL)
{
 if(правило непосредственно применимо к дереву с указателем на
    корень root)
 {
  установка указателей на поддеревья, соответствующие подформулам в
  правиле вывода;
  вывод информации о применяемом правиле и печать поддеревьев;
  преобразование дерева;
  увеличение значения nar на 1;
  вывод новой формулы;
 }
 else
 {
  применение правила к левому поддереву;
  применение правила к правому поддереву;
 }
}
```

Заметим, что правило <u>непосредственно</u> <u>применимо</u> к дереву, если оно применимо ко всему этому дереву (то есть формула, соответствующая дереву, интерпретируется как левая часть некоторого правила переписывания).

Правило <u>косвенно</u> <u>применимо</u> к дереву, если оно непосредственно применимо к некоторому его поддереву, за исключением самого этого дерева.

Правило <u>применимо</u> к дереву, если оно непосредственно или косвенно применимо к нему.

Пример: Правило 1.1 косвенно применимо к дереву, соответствующему формуле a||(b;(c;d)), а именно, к правому его поддереву, корневая вершина которого содержит первый символ предшествования ;.

Таким образом, при применении правила к дереву просматриваются все его поддеревья, пока указатель на корень текущего поддерева, root, не станет равным NULL. То есть, если правило применимо к дереву, оно применяется.

Более подробное описание программы- в приложении, где содержится ее текст с комментариями.

7.Примеры работы программы CANONIC

В этой главе приведены примеры работы программы CANONIC на различных формулах. В процессе работы программы они приводятся к дизъюнкции 1-конъюнкций.

1) Формула $a\overline{\vee}(b;c)$

```
this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
2) -a  *a
3) "a  ~a
4) "(p)  ~(p)
5) a#b  a+b  a!b  a;b
6) a#(p)  a+(p)  a!(p)  a;(p)
7) (p)#a  (p)+a  (p)!a  (p);a
8) (p)#(q)  (p)+(q)  (p)!(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
your formula is:
a#(b;c)
step 1
rule 3.1 is applyed
p=a
q=(b;c)
new formula is:
(a!("(b;c)))+(("a)!(b;c))
rule 4.1 is applyed
p=b
q=c
new formula is:
(a!(("b)!("c)))+(("a)!(b;c))
rule 4.3 is applyed
p=b
new formula is:
(a!((-b)!("c)))+(("a)!(b;c))
rule 4.3 is applyed
p=c
new formula is:
(a!((-b)!(-c)))+(("a)!(b;c))
rule 4.3 is applyed
p=a
new formula is:
(a!((-b)!(-c)))+((-a)!(b;c))
number of applyed rules on step 1 is 5
step 2
rule 1.1 is applyed
p=a
q=(-b)
r=(-c)
new formula is:
((a!(-b))!(-c))+((-a)!(b;c))
number of applyed rules on step 2 is 1
```

```
step 3
number of applyed rules on step 3 is 0
out form is:
((a!(-b))!(-c))+((-a)!(b;c))

2) Формула (a;b);(c;d)

this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
2) -a  *a
3) 'a  ~a
4) '(p)  ~(p)
5) a#b  a+b  a!b  a;b
6) a#(p)  a+(p)  a!(p)  a;(p)
7) (p)#a  (p)+a  (p)!a  (p);a
8) (p)#(q)  (p)+(q)  (p)!(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
your formula is:
(a;b);(c;d)
step 1
rule 1.1 is applyed
p=(a;b)
q=c
r=d
new formula is:
((a;b);c);d
rule 5.1 is applyed
p=a
q=b
r=c
new formula is:
(((a;b)!(b;c))!(a;c));d
number of applyed rules on step 1 is 2
step 2
rule 2.1 is applyed
p=((a;b)!(b;c))
q=(a;c)
r=d
new formula is:
(((a;b)!(b;c));d)!((a;c);d)
rule 5.1 is applyed
p=a
q=c
r=d
new formula is:
(((a;b)!(b;c));d)!(((a;c)!(c;d))!(a;d))
number of applyed rules on step 2 is 2
step 3
rule 1.1 is applyed
p=(((a;b)!(b;c));d)
q=((a;c)!(c;d))
r=(a;d)
```

```
new formula is:
((((a;b)||(b;c));d)||((a;c)||(c;d)))||(a;d)
rule 2.1 is applyed
p=(a;b)
q=(b;c)
r=d
new formula is:
((((a;b);d)||((b;c);d))||((a;c)||(c;d)))||(a;d)
rule 5.1 is applyed
p=a
q=b
r=d
new formula is:
(((((a;b)||(b;d))||(a;d))||((b;c);d))||((a;c)||(c;d)))||(a;d)
rule 5.1 is applyed
p=b
q=c
r=d
new formula is:
(((((a;b)||(b;d))||(a;d))||(((b;c)||(c;d))||(b;d)))||((a;c)||(c;d)))||(a;d)
number of applyed rules on step 3 is 4
step 4
rule 1.1 is applyed
p=((((a;b)||(b;d))||(a;d))||(((b;c)||(c;d))||(b;d)))
q=(a;c)
r=(c;d)
new formula is:
((((((a;b)||(b;d))||(a;d))||(((b;c)||(c;d))||(b;d)))||(a;c))||(c;d))||(a;d)
number of applyed rules on step 4 is 1
step 5
rule 1.1 is applyed
p=(((a;b)||(b;d))||(a;d))
q=((b;c)||(c;d))
r=(b;d)
new formula is:
(((((((a;b)||(b;d))||(a;d))||((b;c)||(c;d)))||(b;d))||(a;c))||(c;d))||(a;d)
number of applyed rules on step 5 is 1
step 6
rule 1.1 is applyed
p=(((a;b)||(b;d))||(a;d))
q=(b;c)
r=(c;d)
new formula is:
((((((((a;b)||(b;d))||(a;d))||(b;c))||(c;d))||(b;d))||(a;c))||(c;d))||(a;d)
number of applyed rules on step 6 is 1
step 7
number of applyed rules on step 7 is 0
out form is:
((((((((a;b)||(b;d))||(a;d))||(b;c))||(c;d))||(b;d))||(a;c))||(c;d))||(a;d)
```

3) Формула $a\|(b\vee(\bar{c};\delta_d))$

```
this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
```

```
2) -a  *a
3) 'a  ~a
4) '(p)  ~(p)
5) a#b  a+b  a!b  a;b
6) a#(p)  a+(p)  a!(p)  a;(p)
7) (p)#a  (p)+a  (p)!a  (p);a
8) (p)#(q)  (p)+(q)  (p)!(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
your formula is:
a!(b+((-c);(*d)))
step 1
rule 2.2 is applyed
p=a
q=b
r=((-c);(*d))
new formula is:
(a!b)+(a!((-c);(*d)))
rule 5.2 is applyed
p=(-c)
q=(*d)
new formula is:
(a!b)+(a!((-c)!(*d)))
number of applyed rules on step 1 is 2
step 2
rule 1.1 is applyed
p=a
q=(-c)
r=(*d)
new formula is:
(a!b)+((a!(-c))!(*d))
number of applyed rules on step 2 is 1
step 3
number of applyed rules on step 3 is 0
out form is:
(a!b)+((a!(-c))!(*d))
```

4) Формула $\tilde{\sqcap}(a \vee (\bar{b};c))$

```
this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
2) -a  *a
3) 'a  ~a
4) '(p)  ~(p)
5) a#b  a+b  a!b  a;b
6) a#(p)  a+(p)  a!(p)  a;(p)
7) (p)#a  (p)+a  (p)!a  (p);a
8) (p)#(q)  (p)+(q)  (p)!(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
```

```
your formula is:
~(a+(b;(-c)))
step 1
rule 4.2 is applyed
p=a
q=(b;(-c))
new formula is:
(~a)+(~(b;(-c)))
rule 4.4 is applyed
p=a
new formula is:
(*a)+(~(b;(-c)))
rule 5.3 is applyed
p=b
q=(-c)
new formula is:
(*a)+(~(b!(-c)))
number of applyed rules on step 1 is 3
step 2
rule 4.1 is applyed
p=b
q=(-c)
new formula is:
(*a)+((~b)!(~(-c)))
rule 4.4 is applyed
p=b
new formula is:
(*a)+((*b)!(~(-c)))
rule 4.4 is applyed
p=(-c)
new formula is:
(*a)+((*b)!(*c))
number of applyed rules on step 2 is 3
step 3
number of applyed rules on step 3 is 0
out form is:
(*a)+((*b)!(*c))
```

5) Формула $(\delta_a;(b;b));\delta_c$

```
this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
2) -a  *a
3) 'a  ~a
4) '(p)  ~(p)
5) a#b  a+b  a!b  a;b
6) a#(p)  a+(p)  a!(p)  a;(p)
7) (p)#a  (p)+a  (p)!a  (p);a
8) (p)#(q)  (p)+(q)  (p)!(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
your formula is:
```

```
((*a);(b;b));(*c)
step 1
rule 1.1 is applyed
p=(*a)
q=b
r=b
new formula is:
(((*a);b);b);(*c)
rule 5.1 is applyed
p=(*a)
q=b
r=b
new formula is:
((((*a);b)¦(b;b))¦((*a);b));(*c)
rule 5.4 is apllyed
p=b
q=b
new formula is:
((((*a);b)¦(*b))¦((*a);b));(*c)
rule 5.5 is applyed
p=(*a)
q=b
new formula is:
((((*a)¦(*b))¦(*b))¦((*a);b));(*c)
rule 5.5 is applyed
p=(*a)
q=b
new formula is:
((((*a)¦(*b))¦(*b))¦((*a)¦(*b)));(*c)
number of applyed rules on step 1 is 5
step 2
rule 1.1 is applyed
p=(((*a)¦(*b))¦(*b))
q=(*a)
r=(*b)
new formula is:
(((((*a)¦(*b))¦(*b))¦(*a))¦(*b));(*c)
rule 2.1 is applyed
p=((((*a)¦(*b))¦(*b))¦(*a))
q=(*b)
r=(*c)
new formula is:
(((((*a)¦(*b))¦(*b))¦(*a));(*c))¦((*b);(*c))
rule 5.5 is applyed
p=(*b)
q=(*c)
new formula is:
(((((*a)¦(*b))¦(*b))¦(*a));(*c))¦((*b)¦(*c))
number of applyed rules on step 2 is 3
step 3
rule 1.1 is applyed
p=(((((*a)¦(*b))¦(*b))¦(*a));(*c))
q=(*b)
r=(*c)
new formula is:
((((((*a)¦(*b))¦(*b))¦(*a));(*c))¦(*b))¦(*c)
rule 2.1 is applyed
p=(((*a)¦(*b))¦(*b))
```

```
q=(*a)
r=(*c)
new formula is:
(((((*a)¦(*b))¦(*b));(*c))¦((*a);(*c)))¦(*b))¦(*c)
rule 5.5 is applyed
p=(*a)
q=(*c)
new formula is:
(((((*a)¦(*b))¦(*b));(*c))¦((*a)¦(*c)))¦(*b))¦(*c)
number of applyed rules on step 3 is 3
step 4
rule 1.1 is applyed
p=(((*a)¦(*b))¦(*b));(*c))
q=(*a)
r=(*c)
new formula is:
((((((*a)¦(*b))¦(*b));(*c))¦(*a))¦(*c))¦(*b))¦(*c)
rule 2.1 is applyed
p=((*a)¦(*b))
q=(*b)
r=(*c)
new formula is:
((((((*a)¦(*b));(*c))¦((*b);(*c)))¦(*a))¦(*c))¦(*b))¦(*c)
rule 5.5 is applyed
p=(*b)
q=(*c)
new formula is:
((((((*a)¦(*b));(*c))¦((*b)¦(*c)))¦(*a))¦(*c))¦(*b))¦(*c)
number of applyed rules on step 4 is 3
step 5
rule 1.1 is applyed
p=(((*a)¦(*b));(*c))
q=(*b)
r=(*c)
new formula is:
(((((((*a)¦(*b));(*c))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
rule 2.1 is applyed
p=(*a)
q=(*b)
r=(*c)
new formula is:
(((((((*a);(*c))¦((*b);(*c)))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
rule 5.5 is applyed
p=(*a)
q=(*c)
new formula is:
(((((((*a)¦(*c))¦((*b);(*c)))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
rule 5.5 is applyed
p=(*b)
q=(*c)
new formula is:
(((((((*a)¦(*c))¦((*b)¦(*c)))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
number of applyed rules on step 5 is 4
step 6
rule 1.1 is applyed
p=((*a)¦(*c))
q=(*b)
r=(*c)
```

```
new formula is:
((((((((*a)¦(*c))¦(*b))¦(*c))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
number of applyed rules on step 6 is 1
step 7
number of applyed rules on step 7 is 0
out form is:
((((((((*a)¦(*c))¦(*b))¦(*c))¦(*b))¦(*c))¦(*a))¦(*c))¦(*b))¦(*c)
```

6) Формула (a‖b);δ_c

```
this program is writed by Tarasyuk I.V.
program CANONIC transforms AFP2-formula
to disjunction of the 1-conjunctions
AFP2-formula may has one of the next forms
1) a
2) -a  *a
3) 'a  ~a
4) '(p)  ~(p)
5) a#b  a+b  a¦b  a;b
6) a#(p)  a+(p)  a¦(p)  a;(p)
7) (p)#a  (p)+a  (p)¦a  (p);a
8) (p)#(q)  (p)+(q)  (p)¦(q)  (p);(q)
where p and q are formulas types 2-8
input formula
sign of end is EOF
formula has been read
your formula is:
(a¦b);(*c)
step 1
rule 2.1 is applyed
p=a
q=b
r=(*c)
new formula is:
(a;(*c))¦(b;(*c))
rule 5.6 is applyed
p=a
q=(*c)
new formula is:
(a¦(*c))¦(b;(*c))
rule 5.6 is applyed
p=b
q=(*c)
new formula is:
(a¦(*c))¦(b¦(*c))
number of applyed rules on step 1 is 3
step 2
rule 1.1 is applyed
p=(a¦(*c))
q=b
r=(*c)
new formula is:
((a¦(*c))¦b)¦(*c)
number of applyed rules on step 2 is 1
step 3
number of applyed rules on step 3 is 0
out form is:
```

((a!(*c))!b)!(*c)

<u>Приложение</u>: <u>Текст</u> <u>программы</u> <u>CANONIC</u> <u>с</u> <u>комментариями</u>

```
/*program CANONIC*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

struct treeel
{
 char treesmb[2];
 struct treeel *father,*son1,*son2;
};
struct chainel
{
 char chainsmb;
 struct chainel *left,*right;
};

typedef struct treeel typevert;
typedef struct chainel typelink;

#define NEWVERT (typevert *)malloc(sizeof(typevert))
#define NEWLINK (typelink *)malloc(sizeof(typelink))

#define ALT '#'
#define DSJ '+'
#define CNC '!'
#define PRC ';'
#define NOC '"'
#define MNO '~'
#define NOT '-'
#define DLT '*'

typevert *ancestor;

main()
{
 int nar,step,letter;
 typelink *currlink,*lastlink,*start,*finish;
 /*nar-число примененных правил на шаге с номером
 step;letter-очередной считываемый
 символ;currlink,lastlink-указатели соответственно на текущее и
 предыдущее звенья цепочки;start,finish-указатели соответственно на
 первое и последнее звенья цепочки*/

 /*предварительная информация и ввод формулы*/
 printf("this program is writed by Tarasyuk I.V.\n");
 printf("program CANONIC transforms AFP2-formula\n");
 printf("to disjunction of the 1-conjunctions\n");
 printf("AFP2-formula may has one of the next forms\n");
 printf("1) a\n");
 printf("2) %ca  %ca\n",NOT,DLT);
 printf("3) %ca  %ca\n",NOC,MNO);
 printf("4) %c(p)  %c(p)\n",NOC,MNO);
```

```
printf("5) a%cb  a%cb  a%cb  a%cb\n",ALT,DSJ,CNC,PRC);
printf("6) a%c(p)  a%c(p)  a%c(p)  a%c(p)\n",ALT,DSJ,CNC,PRC);
printf("7) (p)%ca  (p)%ca  (p)%ca  (p)%ca\n",ALT,DSJ,CNC,PRC);
printf("8) (p)%c(q)  (p)%c(q)  (p)%c(q)  (p)%c(q)\n",ALT,DSJ,CNC,
PRC);
printf("where p and q are formulas types 2-8\n");

printf("input formula\n");
printf("sign of end is EOF\n");

/*преобразование формулы в цепочку*/
currlink=NEWLINK;
lastlink=NEWLINK;
while(((letter=getchar())=='\n')||(letter==' ')||(letter=='\t'))
      ;
currlink->chainsmb=letter;
start=NEWLINK;
start=lastlink=currlink;
while((letter=getchar())!=EOF)
{
 if ((letter!='\n')&&(letter!=' ')&&(letter!='\t'))
 {
  currlink=NEWLINK;
  currlink->left=lastlink;
  lastlink->right=currlink;
  currlink->chainsmb=letter;
  lastlink=currlink;
 }
}
finish=NEWLINK;
finish=currlink;
printf("formula has been read\n");

/*преобразование цепочки в дерево*/
ancestor=NEWVERT;
ancestor->son1=ancestor->son2=NULL;
transfct(start,finish,ancestor);
ancestor=ancestor->son2;
free(ancestor->father);
ancestor->father=NULL;

/*уничтожение цепочки*/
dispchain(start,finish);

/*вывод формулы по дереву*/
printf("your formula is:\n");
writetree(ancestor);printf("\n");

/*применение правил к формуле*/
step=1;
do
{
 printf("step %d\n",step);
 nar=0;
 rule11(ancestor,&nar);
 rule21(ancestor,&nar);
 rule22(ancestor,&nar);
 rule31(ancestor,&nar);
```

```
  rule41(ancestor,&nar);
  rule42(ancestor,&nar);
  rule43(ancestor,&nar);
  rule44(ancestor,&nar);
  rule51(ancestor,&nar);
  rule52(ancestor,&nar);
  rule53(ancestor,&nar);
  rule54(ancestor,&nar);
  rule55(ancestor,&nar);
  rule56(ancestor,&nar);
  printf("number of applyed rules on step %d is %d\n",step,nar);
  step++;
 }
 while(nar!=0);

 /*вывод формулы, полученной в результате применения правил*/
 printf("out form is:\n");
 writetree(ancestor);printf("\n");
}/*end canonic*/

transfct(begin,end,forefath)
/*функция преобразования цепочки в дерево*/

typelink *begin,*end;
typevert *forefath;
/*begin,end-указатели соответственно на первое и последнее звенья
цепочки;forefath-указатель на вершину дерева, к которой нужно
присоединять очередную вершину*/

{
 int depth;
 typelink *car;
 typevert *vertex,*subvert;
 /*depth- разность чисел левых и правых скобок в цепочке; car-
указатель,движущийся по ней; vertex, subvert- указатели
соответственно на вершину-отца и вершину-сына в текущем фрагменте
 дерева*/
 if(begin==end) /*case 1*/
 {
  vertex=NEWVERT;
  vertex->treesmb[0]=begin->chainsmb;
  vertex->treesmb[1]='\0';
  vertex->son1=vertex->son2=NULL;
  vertex->father=forefath;
  if(forefath->son2==NULL)
   forefath->son2=vertex;
  else
   forefath->son1=vertex;
 }
 else if((begin->chainsmb==NOT)||(begin->chainsmb==DLT)) /*case 2*/
 {
  vertex=NEWVERT;
  vertex->treesmb[0]=begin->chainsmb;
  vertex->treesmb[1]=end->chainsmb;
  vertex->son1=vertex->son2=NULL;
  vertex->father=forefath;
  if(forefath->son2==NULL)
```

```
    forefath->son2=vertex;
   else
    forefath->son1=vertex;
  }
  else if(((begin->chainsmb==NOC)||(begin->chainsmb==MNO))&&
          (end->chainsmb!=')')) /*case 3*/
  {
   vertex=NEWVERT;
   vertex->treesmb[0]=begin->chainsmb;
   vertex->treesmb[1]='\0';
   vertex->son1=NULL;
   vertex->father=forefath;
   if(forefath->son2==NULL)
    forefath->son2=vertex;
   else
    forefath->son1=vertex;
   subvert=NEWVERT;
   subvert->treesmb[0]=end->chainsmb;
   subvert->treesmb[1]='\0';
   subvert->son1=subvert->son2=NULL;
   subvert->father=vertex;
   vertex->son2=subvert;
  }
  else if(((begin->chainsmb==NOC)||(begin->chainsmb==MNO))&&
          (end->chainsmb==')')) /*case 4*/
  {
   vertex=NEWVERT;
   vertex->treesmb[0]=begin->chainsmb;
   vertex->treesmb[1]='\0';
   vertex->son1=vertex->son2=NULL;
   vertex->father=forefath;
   if(forefath->son2==NULL)
    forefath->son2=vertex;
   else
    forefath->son1=vertex;
   transfct(begin->right->right,end->left,vertex);
  }
  else if((begin->chainsmb!='(')&&(end->chainsmb!=')')) /*case 5*/
  {
   vertex=NEWVERT;
   vertex->treesmb[0]=begin->right->chainsmb;
   vertex->treesmb[1]='\0';
   vertex->father=forefath;
   if(forefath->son2==NULL)
    forefath->son2=vertex;
   else
    forefath->son1=vertex;
   subvert=NEWVERT;
   subvert->treesmb[0]=end->chainsmb;
   subvert->treesmb[1]='\0';
   subvert->son1=subvert->son2=NULL;
   subvert->father=vertex;
   vertex->son2=subvert;
   subvert=NEWVERT;
   subvert->treesmb[0]=begin->chainsmb;
   subvert->treesmb[1]='\0';
   subvert->son1=subvert->son2=NULL;
   subvert->father=vertex;
```

```
  vertex->son1=subvert;
 }
 else if(begin->chainsmb!='(') /*case 6*/
 {
  vertex=NEWVERT;
  vertex->treesmb[0]=begin->right->chainsmb;
  vertex->treesmb[1]='\0';
  vertex->son2=NULL;
  vertex->father=forefath;
  if(forefath->son2==NULL)
   forefath->son2=vertex;
  else
   forefath->son1=vertex;
  subvert=NEWVERT;
  subvert->treesmb[0]=begin->chainsmb;
  subvert->treesmb[1]='\0';
  subvert->son1=subvert->son2=NULL;
  subvert->father=vertex;
  vertex->son1=subvert;
  transfct(begin->right->right->right,end->left,vertex);
 }
 else if(end->chainsmb!=')') /*case 7*/
 {
  vertex=NEWVERT;
  vertex->treesmb[0]=end->left->chainsmb;
  vertex->treesmb[1]='\0';
  vertex->son1=NULL;
  vertex->father=forefath;
  if(forefath->son2==NULL)
   forefath->son2=vertex;
  else
   forefath->son1=vertex;
  subvert=NEWVERT;
  subvert->treesmb[0]=end->chainsmb;
  subvert->treesmb[1]='\0';
  subvert->son1=subvert->son2=NULL;
  subvert->father=vertex;
  vertex->son2=subvert;
  transfct(begin->right,end->left->left->left,vertex);
 }
 else /*case 8*/
 {
  car=NEWLINK;
  car=begin->right;
  for(depth=1;depth!=0;car=car->right)
  {
   if(car->chainsmb=='(') depth++;
   if(car->chainsmb==')') depth--;
  }
 vertex=NEWVERT;
 vertex->treesmb[0]=car->chainsmb;
 vertex->treesmb[1]='\0';
 vertex->son1=vertex->son2=NULL;
 vertex->father=forefath;
 if(forefath->son2==NULL)
  forefath->son2=vertex;
 else
  forefath->son1=vertex;
```

```
  transfct(car->right->right,end->left,vertex);
  transfct(begin->right,car->left->left,vertex);
  }
 }/*end transfct*/

 dispchain(begin,end)
 /*функция уничтожения цепочки*/

 typelink *begin,*end;
 /*begin,end -указатели соответственно на начало и конец цепочки*/

 {
  typelink *car;
  /*car -указатель, перемещающийся по цепочке*/

  car=NEWLINK;
  car=end;
  if(begin!=end)
  {
   do
   {
    car=car->left;
    free(car->right);
   }
   while(car!=begin);
  }
  free(car);
 }/*end dispchain*/

writetree(root)
/*функция печати формулы, преобразованной в дерево*/

typevert *root;
/*root-указатель на корень дерева*/

{
 if(root!=NULL)
 {
  if(!(isalpha(root->treesmb[0]))&&(root!=ancestor)) putchar('(');
  writetree(root->son1);
  putchar(root->treesmb[0]);
  if(root->treesmb[1]!='\0') putchar(root->treesmb[1]);
  writetree(root->son2);
  if(!(isalpha(root->treesmb[0]))&&(root!=ancestor)) putchar(')');
 }
}/*end writetree*/

copytree(young,old)
/*функция копирования дерева*/

typevert *young,*old;
/*old-указатель на корень дерева, с которого делается копия с
соответствующим указателем young*/

{
 typevert *s1,*s2;
 /*s1,s2-указатели соответственно на первого и второго сыновей
копируемой вершины нового дерева*/
```

```
  young->treesmb[0]=old->treesmb[0];
  young->treesmb[1]=old->treesmb[1];
  if(old->son1!=NULL)
  {
   s1=NEWVERT;
   s1->father=young;
   young->son1=s1;
   copytree(s1,old->son1);
  }
  else young->son1=NULL;
  if(old->son2!=NULL)
  {
   s2=NEWVERT;
   s2->father=young;
   young->son2=s2;
   copytree(s2,old->son2);
  }
  else young->son2=NULL;
 }/*end copytree*/

int leaf(root)
/*функция, определяющая, является ли дерево листом*/

typevert *root;
/*root-указатель на корень дерева*/
{
 if((root->treesmb[0]==NOT)||(root->treesmb[0]==DLT)||
    isalpha(root->treesmb[0]))
  return 1;
 else
  return 0;
}/*end leaf*/

rule11(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*r;

 if(root!=NULL)
 {
  if(((root->treesmb[0]==PRC)&&(root->son2->treesmb[0]==PRC))||
     ((root->treesmb[0]==CNC)&&(root->son2->treesmb[0]==CNC))||
     ((root->treesmb[0]==DSJ)&&(root->son2->treesmb[0]==DSJ)))
  {
   p=NEWVERT;
   q=NEWVERT;
   r=NEWVERT;
   p=root->son1;
   q=root->son2->son1;
   r=root->son2->son2;

   printf("rule 1.1 is applyed\n");
   printf("p=");writetree(p);printf("\n");
```

```
    printf("q=");writetree(q);printf("\n");
    printf("r=");writetree(r);printf("\n");

    root->son1=q->father;
    root->son2=p;

    q->father->son1=r;
    q->father->son2=q;

    r->father=root;
    root->son2=r;
    p->father=q->father;
    q->father->son1=p;

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule11(root->son1,addrnar);
    rule11(root->son2,addrnar);
   }
  }
}/*end rule11*/

rule21(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*r,*next;
 char ch;

 if(root!=NULL)
 {
  if(((root->treesmb[0]==PRC)&&(root->son1->treesmb[0]==CNC))||
     ((root->treesmb[0]==PRC)&&(root->son1->treesmb[0]==DSJ))||
     ((root->treesmb[0]==CNC)&&(root->son1->treesmb[0]==DSJ)))
  {
   p=NEWVERT;
   q=NEWVERT;
   r=NEWVERT;
   p=root->son1->son1;
   q=root->son1->son2;
   r=root->son2;

   printf("rule 2.1 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");
   printf("r=");writetree(r);printf("\n");

   ch=root->treesmb[0];
   root->treesmb[0]=root->son1->treesmb[0];
   root->son1->treesmb[0]=ch;

   next=NEWVERT;
   next->treesmb[0]=ch;
```

```
    next->treesmb[1]='\0';
    next->father=root;
    root->son2=next;
    r->father=next;
    next->son2=r;

    q->father=next;
    next->son1=q;

    next=NEWVERT;
    copytree(next,r);
    next->father=p->father;
    p->father->son2=next;

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule21(root->son1,addrnar);
    rule21(root->son2,addrnar);
   }
 }
}/*end rule21*/

rule22(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*r,*next;
 char ch;
 if(root!=NULL)
 {
  if(((root->treesmb[0]==PRC)&&(root->son2->treesmb[0]==CNC))||
     ((root->treesmb[0]==PRC)&&(root->son2->treesmb[0]==DSJ))||
     ((root->treesmb[0]==CNC)&&(root->son2->treesmb[0]==DSJ)))
  {
   p=NEWVERT;
   q=NEWVERT;
   r=NEWVERT;
   p=root->son1;
   q=root->son2->son1;
   r=root->son2->son2;

   printf("rule 2.2 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");
   printf("r=");writetree(r);printf("\n");

   ch=root->treesmb[0];
   root->treesmb[0]=root->son2->treesmb[0];
   root->son2->treesmb[0]=ch;

   next=NEWVERT;
   next->treesmb[0]=ch;
```

```
    next->treesmb[1]='\0';
    next->father=root;
    root->son1=next;
    p->father=next;
    next->son1=p;

    q->father=next;
    next->son2=q;

    next=NEWVERT;
    copytree(next,p);
    next->father=r->father;
    r->father->son1=next;

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule22(root->son1,addrnar);
    rule22(root->son2,addrnar);
   }
 }
}/*end rule22*/

rule31(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*vertex,*next;

 if(root!=NULL)
 {
  if(root->treesmb[0]==ALT)
  {
   p=NEWVERT;
   q=NEWVERT;
   p=root->son1;
   q=root->son2;

   printf("rule 3.1 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");

   root->treesmb[0]=CNC;

   next=NEWVERT;
   next->treesmb[0]=NOC;
   next->treesmb[1]='\0';
   next->father=root;
   root->son2=next;
   q->father=next;
   next->son2=q;
   next->son1=NULL;

   vertex=NEWVERT;
```

```
    vertex->treesmb[0]=DSJ;
    vertex->treesmb[1]='\0';
    vertex->father=root->father;
    if(root->father!=NULL)
    {
     if(root->father->son2==root)
       root->father->son2=vertex;
     else
       root->father->son1=vertex;
    }
    else
     ancestor=vertex;
    vertex->son1=root;
    root->father=vertex;

    next=NEWVERT;
    next->treesmb[0]=CNC;
    next->treesmb[1]='\0';
    next->father=vertex;
    vertex->son2=next;
    next=NEWVERT;
    next->treesmb[0]=NOC;
    next->treesmb[1]='\0';
    next->father=vertex->son2;
    vertex->son2->son1=next;
    next->son1=NULL;

    next=NEWVERT;
    copytree(next,p);
    next->father=vertex->son2->son1;
    vertex->son2->son1->son2=next;

    next=NEWVERT;
    copytree(next,q);
    next->father=vertex->son2;
    vertex->son2->son2=next;
    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule31(root->son1,addrnar);
    rule31(root->son2,addrnar);
   }
  }
}/*end rule31*/

rule41(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*vertex,*next;
 char ch;
```

```
if(root!=NULL)
{
 if(((root->treesmb[0]==NOC)||(root->treesmb[0]==MNO))&&
    ((root->son2->treesmb[0]==CNC)||(root->son2->treesmb[0]==PRC)))
 {
  p=NEWVERT;
  q=NEWVERT;
  p=root->son2->son1;
  q=root->son2->son2;

  printf("rule 4.1 is applyed\n");
  printf("p=");writetree(p);printf("\n");
  printf("q=");writetree(q);printf("\n");

  vertex=NEWVERT;
  vertex=p->father;
  vertex->treesmb[0]=CNC;

  ch=root->treesmb[0];
  vertex->father=root->father;
  if(root->father!=NULL)
  {
   if(root->father->son2==root)
    root->father->son2=vertex;
   else
    root->father->son1=vertex;
  }
  else
   ancestor=vertex;
  free(root);

  next=NEWVERT;
  next->treesmb[0]=ch;
  next->treesmb[1]='\0';
  next->father=vertex;
  vertex->son1=next;
  next->son2=p;
  p->father=next;
  next->son1=NULL;

  next=NEWVERT;
  next->treesmb[0]=ch;
  next->treesmb[1]='\0';
  next->father=vertex;
  vertex->son2=next;
  next->son2=q;
  q->father=next;
  next->son1=NULL;

  (*addrnar)++;
  printf("new formula is:\n");
  writetree(ancestor);printf("\n");
 }
 else
 {
  rule41(root->son1,addrnar);
  rule41(root->son2,addrnar);
 }
```

```
 }
}/*end rule41*/

rule42(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*vertex,*next;
 char ch;

 if(root!=NULL)
 {
  if(((root->treesmb[0]==NOC)||(root->treesmb[0]==MNO))&&
      (root->son2->treesmb[0]==DSJ))
   {
    p=NEWVERT;
    q=NEWVERT;
    p=root->son2->son1;
    q=root->son2->son2;

    printf("rule 4.2 is applyed\n");
    printf("p=");writetree(p);printf("\n");
    printf("q=");writetree(q);printf("\n");

    vertex=NEWVERT;
    vertex=p->father;
    ch=root->treesmb[0];
    vertex->father=root->father;
    if(root->father!=NULL)
    {
     if(root->father->son2==root)
      root->father->son2=vertex;
     else
      root->father->son1=vertex;
    }
    else
     ancestor=vertex;
    free(root);

    next=NEWVERT;
    next->treesmb[0]=ch;
    next->treesmb[1]='\0';
    next->father=vertex;
    vertex->son1=next;
    next->son2=p;
    p->father=next;
    next->son1=NULL;

    next=NEWVERT;
    next->treesmb[0]=ch;
    next->treesmb[1]='\0';
    next->father=vertex;
    vertex->son2=next;
    next->son2=q;
    q->father=next;
    next->son1=NULL;
```

```
    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule42(root->son1,addrnar);
    rule42(root->son2,addrnar);
   }
  }
 }/*end rule42*/

rule43(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p;
 char ch;

 if(root!=NULL)
 {
  if((root->treesmb[0]==NOC)&&leaf(root->son2))
  {
   p=NEWVERT;
   p=root->son2;

   printf("rule 4.3 is applyed\n");
   printf("p=");writetree(p);printf("\n");

   p->father=root->father;
   if(root->father!=NULL)
   {
    if(root->father->son2==root)
     root->father->son2=p;
    else
     root->father->son1=p;
   }
   else
    ancestor=p;
   free(root);

   if(p->treesmb[0]!=NOT)
   {
    if(p->treesmb[0]==DLT)
     ch=p->treesmb[1];
    else
     ch=p->treesmb[0];
    p->treesmb[0]=NOT;
    p->treesmb[1]=ch;
   }

   (*addrnar)++;
   printf("new formula is:\n");
   writetree(ancestor);printf("\n");
  }
  else
  {
```

```
    rule43(root->son1,addrnar);
    rule43(root->son2,addrnar);
   }
  }
 }/*end rule43*/

 rule44(root,addrnar)
 typevert *root;
 int *addrnar;

 {
  typevert *p;
  char ch;

  if(root!=NULL)
  {
   if((root->treesmb[0]==MNO)&&leaf(root->son2))
   {
    p=NEWVERT;
    p=root->son2;

    printf("rule 4.4 is applyed\n");
    printf("p=");writetree(p);printf("\n");

    p->father=root->father;
    if(root->father!=NULL)
    {
     if(root->father->son2==root)
      root->father->son2=p;
     else
      root->father->son1=p;
    }
    else
     ancestor=p;
    free(root);

    if(p->treesmb[0]!=DLT)
    {
     if(p->treesmb[0]==NOT)
      ch=p->treesmb[1];
     else
      ch=p->treesmb[0];
     p->treesmb[0]=DLT;
     p->treesmb[1]=ch;
    }

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
  else
  {
   rule44(root->son1,addrnar);
   rule44(root->son2,addrnar);
  }
 }
}/*end rule44*/
```

```
rule51(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q,*r,*vertex,*next;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&(root->son1->treesmb[0]==PRC)&&
     leaf(root->son1->son1)&&leaf(root->son1->son2)&&
     leaf(root->son2))
  {
   p=NEWVERT;
   q=NEWVERT;
   r=NEWVERT;
   p=root->son1->son1;
   q=root->son1->son2;
   r=root->son2;

   printf("rule 5.1 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");
   printf("r=");writetree(r);printf("\n");

   root->treesmb[0]=CNC;

   next=NEWVERT;
   next->treesmb[0]=PRC;
   next->treesmb[1]='\0';
   next->father=root;
   root->son2=next;
   next->son2=r;
   r->father=next;

   next=NEWVERT;
   copytree(next,q);
   next->father=r->father;
   r->father->son1=next;

   vertex=NEWVERT;
   vertex->treesmb[0]=CNC;
   vertex->treesmb[1]='\0';
   vertex->father=root->father;
   if(root->father!=NULL)
   {
    if(root->father->son2==root)
     root->father->son2=vertex;
    else
     root->father->son1=vertex;
   }
   else
    ancestor=vertex;
   vertex->son1=root;
   root->father=vertex;

   next=NEWVERT;
   next->treesmb[0]=PRC;
```

```
    next->treesmb[1]='\0';
    next->father=vertex;
    vertex->son2=next;

    next=NEWVERT;
    copytree(next,p);
    next->father=vertex->son2;
    vertex->son2->son1=next;

    next=NEWVERT;
    copytree(next,r);
    next->father=vertex->son2;
    vertex->son2->son2=next;

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule51(root->son1,addrnar);
    rule51(root->son2,addrnar);
   }
  }
}/*end rule51*/

rule52(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&(root->son1->treesmb[0]==NOT)&&
      leaf(root->son2))
  {
   p=NEWVERT;
   q=NEWVERT;
   p=root->son1;
   q=root->son2;

   printf("rule 5.2 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");

   root->treesmb[0]=CNC;

   (*addrnar)++;
   printf("new formula is:\n");
   writetree(ancestor);printf("\n");
  }
  else
  {
   rule52(root->son1,addrnar);
   rule52(root->son2,addrnar);
  }
```

```
  }
 }/*end rule52*/

rule53(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&(root->son2->treesmb[0]==NOT)&&
     leaf(root->son1))
  {
   p=NEWVERT;
   q=NEWVERT;
   p=root->son1;
   q=root->son2;

   printf("rule 5.3 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");

   root->treesmb[0]=CNC;

   (*addrnar)++;
   printf("new formula is:\n");
   writetree(ancestor);printf("\n");
  }
  else
  {
   rule53(root->son1,addrnar);
   rule53(root->son2,addrnar);
  }
 }
}/*end rule53*/

rule54(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&
     isalpha(root->son1->treesmb[0])&&isalpha(root->son2->
     treesmb[0])&&
     (root->son1->treesmb[0]==root->son2->treesmb[0]))
  {
   p=NEWVERT;
   q=NEWVERT;
   p=root->son1;
   q=root->son2;

   printf("rule 5.4 is apllyed\n");
```

```
    printf("p=");writetree(p);printf("\n");
    printf("q=");writetree(q);printf("\n");

    root->treesmb[0]=DLT;
    root->treesmb[1]=p->treesmb[0];

    root->son1=root->son2=NULL;
    free(p);
    free(q);

    (*addrnar)++;
    printf("new formula is:\n");
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule54(root->son1,addrnar);
    rule54(root->son2,addrnar);
   }
  }
}/*end rule54*/

rule55(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q;
 char ch;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&(root->son1->treesmb[0]==DLT)&&
     leaf(root->son2))
   {
    p=NEWVERT;
    q=NEWVERT;
    p=root->son1;
    q=root->son2;

    printf("rule 5.5 is applyed\n");
    printf("p=");writetree(p);printf("\n");
    printf("q=");writetree(q);printf("\n");

    root->treesmb[0]=CNC;

    if(q->treesmb[0]!=DLT)
    {
     if(q->treesmb[0]==NOT)
      ch=q->treesmb[1];
     else
      ch=q->treesmb[0];
     q->treesmb[0]=DLT;
     q->treesmb[1]=ch;
    }

    (*addrnar)++;
    printf("new formula is:\n");
```

```
    writetree(ancestor);printf("\n");
   }
   else
   {
    rule55(root->son1,addrnar);
    rule55(root->son2,addrnar);
   }
  }
}/*end rule55*/

rule56(root,addrnar)
typevert *root;
int *addrnar;

{
 typevert *p,*q;

 if(root!=NULL)
 {
  if((root->treesmb[0]==PRC)&&(root->son2->treesmb[0]==DLT)&&
     leaf(root->son1))
  {
   p=NEWVERT;
   q=NEWVERT;
   p=root->son1;
   q=root->son2;

   printf("rule 5.6 is applyed\n");
   printf("p=");writetree(p);printf("\n");
   printf("q=");writetree(q);printf("\n");

   root->treesmb[0]=CNC;

   (*addrnar)++;
   printf("new formula is:\n");
   writetree(ancestor);printf("\n");
  }
  else
  {
   rule56(root->son1,addrnar);
   rule56(root->son2,addrnar);
  }
 }
}/*end rule56*/
```

И. В. Тарасюк

Литература:

[Kot78] Kotov,V.E.:An algebra for parallelism based on Petri nets.LNCS,Vol 64,p.39-55,1978.

[P81] Peterson,J.L.:Petri net theory and modelling of systems.Prentice Hall,1981.(Имеется перевод на русский язык. Дж. Питерсон: Теория сетей Петри и моделирование систем. М., Мир, 1984).

[Ch89] Cherkasova,L.A.:Posets with non-actions:A model for concurrent nondeterministic processes.Arbeitspapiere der GMD,№ 403,1989.

[Ch90-1] Cherkasova,L.A.:Algebra AFP_2 for concurrent nondeterministic processes:Fully abstract model and complete axiomatization.Reihe Informatic,№ 72,1990.

[Ch90-2] Cherkasova,L.A.:A fully abstract model for concurrent nondeterministic processes based on posets with non-actions.Computer science/Department of software technology,Report CS-R 9031,1990.

Оглавление: