Интерпретатор модельного языка взаимодействующих процессов

Вариант Г

Олег Овласюк

Группа 207

***Постановка задания***

*Синтаксис языка*

*program ::= process\_expression*

*process\_expression ::= sequence { "||" sequence }*

*sequence ::= term { ";" term }*

*term ::= process | "(" process\_expression ")"*

*process ::= process\_name ":" process\_description*

*process\_name ::= id*

*process\_description ::= "loop" { reaction } [default\_reaction]*

*reaction ::= "when" event "::" { statement }*

*default\_reaction ::= “else” “::” { statement }*

*event ::= message*

*message ::= id*

*statement ::= send | print | die | children*

*send ::= process\_name "<-" message ";"*

*print ::= "print" string ";" | "println" ";"*

*die ::= "die" ";"*

*children ::= "(" process\_expression ")" ";"*

Идентификаторы (id) - это непустые последовательности латинских букв, цифр и символов подчеркивания, начинающихся с латинской буквы, не являющиеся ключевыми словами (они написаны в грамматике в двойных кавычках), строковые константы (string) - это помещенные в двойные кавычки последовательности символов, в которых не встречается двойная кавычка.

*Контекстные условия*

После успешного синтаксического анализа семантический анализатор должен проверить выполнение всех следующих контекстных условий:

* Все определяемые процессы должны иметь различные имена.
* Запрещается использовать в конструкции отправки сообщения имя процесса, которое не определено (процесс с таким именем не определен в программе).
* В каждой последовательности statement может использоваться не более одного die, причем если он используется, то он должен быть последним statement'ом в этой последовательности.
* Все сообщения, входящие в конструкции reaction после ключевого слова “when” одного и того же процесса, должны быть разными.

*Семантика языка*

Интерпретация программы состоит из двух частей:

1. создание процессов.
2. параллельное выполнение процессов.

Создание процессов - это заведение структур, описывающих каждый из процессов. При этом ни один из процессов не начинает выполняться, пока не будут созданы все процессы.

Параллельное выполнение процессов. Каждый процесс работает по шагам. Все шаги состоят из следующих частей:

1. получение очередного сообщения для обработки (если очередь сообщений непуста).
2. определение соответствующего reaction для выбранного сообщения.
3. выполнение reaction.

Каждый процесс содержит очередь входящих сообщений. Вначале все очереди всех процессов пусты. Любой процесс может послать сообщение любому процессу - тогда сообщение будет помещено в очередь входящих сообщений процесса-получателя.

Если входящая очередь сообщений пуста и отсутствует default\_reaction, процесс блокируется до тех пор, пока в очереди не появится хотя бы одно сообщение. Если входящая очередь сообщений пуста, но есть default\_reaction, то выполняется default\_reaction. Если входящая очередь сообщений непуста, из нее извлекается очередное сообщение.

Определение соответствующего reaction. Если для этого сообщения есть reaction с упоминанием этого сообщения после when, то выполняется этот reaction. Если для сообщения, изъятого из очереди сообщений, нет reaction с этим же сообщением после when и есть default\_reaction, выполняется default\_reaction (“сообщение теряется”). В противном случае считается, что происходит ошибка времени выполнения (“процессу послали сообщение, которое он не умеет обрабатывать”), интерпретатор завершает свою работу с кодом возврата, соответствующим ошибке времени выполнения (см.ниже), о чем делается запись в протоколе с указанием причины (см.ниже).

Выполнение reaction. Последовательность операторов (statement’ов) выполняется по порядку, как они указаны в программе. Следующий оператор не начинает выполняться, пока не закончил выполняться предыдущий. Оператор печати (print/println) печатает указанный у него текст на стандартный вывод (println переводит вывод на новую строку). Оператор отправки сообщения помещает указанное сообщение в очередь указанному процессу. Операция отправки сообщения не является блокирующей (т.е. не надо ждать, пока это сообщение будет прочитано процессом-получателем, но операция отправки сообщения считается завершенной тогда, когда сообщение поместилось в очередь процесса-получателя). Оператор завершения процесса die завершает выполнение процесса. Любая попытка отправки сообщения завершенному процессу приводит к ошибке времени выполнения.

Интерпретатор корректно завершает свою работу, когда умирают все запущенные им процессы.

***Аналитическая часть***

*Диаграмма состояний лексического анализатора*

**

*Трудности применения РС-метода к грамматике*

***Исходная грамматика:***

*program ::= process\_expression*

*process\_expression ::= sequence { "||" sequence }*

*sequence ::= term { ";" term }*

*term ::= process | "(" process\_expression ")"*

*process ::= process\_name ":" process\_description*

*process\_name ::= id*

*process\_description ::= "loop" { reaction } [default\_reaction]*

*reaction ::= "when" event "::" { statement }*

*default\_reaction ::= “else” “::” { statement }*

*event ::= message*

*message ::= id*

*statement ::= send | print | die | children*

*send ::= process\_name "<-" message ";"*

*print ::= "print" string ";" | "println" ";"*

*die ::= "die" ";"*

*children ::= process\_expression ";"*

К грамматике в исходном виде не применим РС-метод из-за неоднозначности. После считывания процесса, описанного в операторе children, считывается точка с запятой, которая может означать либо конец term из sequence, либо конец оператора children. Если следом идет последовательность из term, то разрешить неоднозначность ресурсозатратно, так как это разрешается только в конце этой последовательности (которая может быть очень длинной).

Решение проблемы - изменение языка следующим образом:

*children ::= "(" process\_expression ")" ";"*

*Построение интерпретатора*

***Промежуточное представление:***

Класс Program содержит контейнер объектов класса Process и список начальных процессов.

Класс Process содержит имя процесса, контейнер объектов класса Event и список процессов, которые нужно запустить после данного процесса.

Класс Event содержит имя события и последовательность объектов класса Statement.

Класс Statement содержит тип стэйтмента и (не всегда) значение.

Типы Statement:

* PRINT
* PRINTLN
* NEW
* SEND
* DIE
* ID (со значением)
* STRING (со значением)

***Эксперименты***

*Пинг-понг процессов*

Starter : loop else :: print “Yahoo, we are starting!”; println;

 A <- start; die;

||

A : loop

when start :: print “A sends ping to B”; println;

 B <- pingB1;

when pongA2 :: print “A receives pong from B”; println;

 print “A sends ping to B”; println;

 B <- pingB2;

when pongA3 :: print “A receives pong from B”; println;

 print “Finishing A”; println;

 die;

||

B : loop

when pingB1 :: print “B receives ping from A”; println;

 print “B sends pong to A”; println;

 A <- pongA2;

when pingB2 :: print “B receives ping from A”; println;

 print “B send pong to A”; println;

 A <- pongA3;

 print “Finishing B”; println;

 die;

***Вывод программы:***

|  |  |
| --- | --- |
| Запуск 1: | Запуск 2: |
| Yahoo, we are starting!A sends ping to BB recieves ping from AB sends pong to AA recieves pong from BA sends ping to BB recieves ping from AB send pong to AA recieves pong from B Finishing BFinishing A | Yahoo, we are starting!A sends ping to BB recieves ping from AB sends pong to AA recieves pong from BA sends ping to BB recieves ping from AB send pong to AA recieves pong from B Finishing AFinishing B |

***Файл логов:***

Запуск 1:

======= start tracing =======

[Tue May 28 03:41:12 2013] start executing

[Tue May 28 03:41:12 2013] Process 'Starter' is created

[Tue May 28 03:41:12 2013] Process 'A' is created

[Tue May 28 03:41:12 2013] Process 'B' is created

[Tue May 28 03:41:12 2013] Message 'start' is recieved

[Tue May 28 03:41:12 2013] Message 'start' is sent to 'A'

[Tue May 28 03:41:12 2013] process 'Starter' is died

[Tue May 28 03:41:12 2013] Message 'pingB1' is recieved

[Tue May 28 03:41:12 2013] Message 'pingB1' is sent to 'B'

[Tue May 28 03:41:12 2013] Message 'pongA2' is recieved

[Tue May 28 03:41:12 2013] Message 'pongA2' is sent to 'A'

[Tue May 28 03:41:12 2013] Message 'pingB2' is sent to 'B'

[Tue May 28 03:41:12 2013] Message 'pingB2' is recieved

[Tue May 28 03:41:12 2013] Message 'pongA3' is recieved

[Tue May 28 03:41:12 2013] Message 'pongA3' is sent to 'A'

[Tue May 28 03:41:12 2013] process 'B' is died

[Tue May 28 03:41:12 2013] process 'A' is died

[Tue May 28 03:41:12 2013] terminating...

======= end of tracing =======

Запуск 2:

======= start tracing =======

[Tue May 28 03:43:26 2013] start executing

[Tue May 28 03:43:26 2013] Process 'Starter' is created

[Tue May 28 03:43:26 2013] Process 'A' is created

[Tue May 28 03:43:26 2013] Process 'B' is created

[Tue May 28 03:43:26 2013] Message 'start' is recieved

[Tue May 28 03:43:26 2013] Message 'start' is sent to 'A'

[Tue May 28 03:43:26 2013] process 'Starter' is died

[Tue May 28 03:43:26 2013] Message 'pingB1' is recieved

[Tue May 28 03:43:26 2013] Message 'pingB1' is sent to 'B'

[Tue May 28 03:43:26 2013] Message 'pongA2' is recieved

[Tue May 28 03:43:26 2013] Message 'pongA2' is sent to 'A'

[Tue May 28 03:43:26 2013] Message 'pingB2' is sent to 'B'

[Tue May 28 03:43:26 2013] Message 'pingB2' is recieved

[Tue May 28 03:43:26 2013] Message 'pongA3' is recieved

[Tue May 28 03:43:26 2013] Message 'pongA3' is sent to 'A'

[Tue May 28 03:43:26 2013] process 'A' is died

[Tue May 28 03:43:26 2013] process 'B' is died

[Tue May 28 03:43:26 2013] terminating...

======= end of tracing =======

*Перегонки*

Starter : loop

else :: A <-start; B <-start; die;

||

A : loop

when start :: R <- fromA;

when win :: print "A: i'm Winner"; println; die;

when kill :: die;

||

B : loop

when start :: R <- fromB;

when win :: print "B: i'm Winner"; println; die;

when kill :: die;

||

R : loop

when fromA :: A <- win; B <-kill; die;

when fromB :: B <- win; A <-kill; die;

***Вывод программы:***

|  |  |
| --- | --- |
| Запуск 1: | Запуск 2: |
| A: i'm Winner | B: i'm Winner |

***Файл логов:***

Запуск 1:

======= start tracing =======

[Tue May 28 04:00:57 2013] start executing

[Tue May 28 04:00:57 2013] Process 'Starter' is created

[Tue May 28 04:00:57 2013] Process 'B' is created

[Tue May 28 04:00:57 2013] Process 'R' is created

[Tue May 28 04:00:57 2013] Message 'start' is sent to 'A'

[Tue May 28 04:00:57 2013] Process 'A' is created

[Tue May 28 04:00:57 2013] Message 'start' is sent to 'B'

[Tue May 28 04:00:57 2013] process 'Starter' is died

[Tue May 28 04:00:57 2013] Message 'start' is recieved

[Tue May 28 04:00:57 2013] Message 'fromA' is sent to 'R'

[Tue May 28 04:00:57 2013] Message 'start' is recieved

[Tue May 28 04:00:57 2013] Message 'fromB' is sent to 'R'

[Tue May 28 04:00:57 2013] Message 'fromA' is recieved

[Tue May 28 04:00:57 2013] Message 'win' is sent to 'A'

[Tue May 28 04:00:57 2013] Message 'win' is recieved

[Tue May 28 04:00:57 2013] Message 'kill' is sent to 'B'

[Tue May 28 04:00:57 2013] process 'A' is died

[Tue May 28 04:00:57 2013] Message 'kill' is recieved

[Tue May 28 04:00:57 2013] process 'B' is died

[Tue May 28 04:00:57 2013] process 'R' is died

[Tue May 28 04:00:57 2013] terminating...

======= end of tracing =======

Запуск 2:

======= start tracing =======

[Tue May 28 04:04:27 2013] start executing

[Tue May 28 04:04:27 2013] Process 'Starter' is created

[Tue May 28 04:04:27 2013] Process 'A' is created

[Tue May 28 04:04:27 2013] Process 'B' is created

[Tue May 28 04:04:27 2013] Process 'R' is created

[Tue May 28 04:04:27 2013] Message 'start' is sent to 'A'

[Tue May 28 04:04:27 2013] Message 'start' is sent to 'B'

[Tue May 28 04:04:27 2013] Message 'start' is recieved

[Tue May 28 04:04:27 2013] Message 'start' is recieved

[Tue May 28 04:04:27 2013] Message 'fromB' is sent to 'R'

[Tue May 28 04:04:27 2013] process 'Starter' is died

[Tue May 28 04:04:27 2013] Message 'fromA' is sent to 'R'

[Tue May 28 04:04:27 2013] Message 'fromB' is recieved

[Tue May 28 04:04:27 2013] Message 'win' is sent to 'B'

[Tue May 28 04:04:27 2013] Message 'win' is recieved

[Tue May 28 04:04:27 2013] Message 'kill' is sent to 'A'

[Tue May 28 04:04:27 2013] process 'B' is died

[Tue May 28 04:04:27 2013] process 'R' is died

[Tue May 28 04:04:27 2013] Message 'kill' is recieved

[Tue May 28 04:04:27 2013] process 'A' is died

[Tue May 28 04:04:27 2013] terminating...

======= end of tracing =======