Лекция 20-10-04

Параллельные процессы. Каналы.

Тема этой лекции – моделирование и спецификация систем с параллельными процессами. Одна из целей лекции – показать, что модели нетривиальных систем можно рассматривать как простой объект, которым можно манипулировать так же, как и с алгебраическими формулами. Если у нас есть сложная формула, то ее можно упростить, и только потом производить вычисления. Для программ можно привести следующую иллюстрацию:

X:=10; … /* X не используется */; X:=12;
Очевидно, что такую программу можно упростить:
… /* X не используется */; X:=12;
или
X:=12; … /* X не используется */;

Когда мы рассматривали доказательства, то уже делали некоторые преобразования программ. Эти преобразования были не тождественными, но сохраняли нужные нам свойства.

Сегодня мы посмотрим, как RSL позволяет моделировать системы с параллельными вычислениями. Введем некоторые новые понятия. В языке не вводится понятие «процесс». Вычислимая часть спецификации – это одно выражение, из которого выделяются активности, происходящие одновременно.

выражение = ^^^^^ || ^^^^^ || ^^^^^

|| - комбинатор параллельности, его можно трактовать как бинарный оператор. Он ассоциативен и коммутативен. Единственное условие, которое накладывается на его операнды, - они не должны вычислять никакого значения, т.е. должны возвращать тип Unit.

Как происходит координация, синхронизация и блокировка в параллельных системах?

Рассмотрим пример:
(x:=1; y:=x) || (x:=2; y:=x)
Что будет вычислено в результате?
Правило: Если в выражениях, разделенных комбинатором параллельности, модифицируется хотя бы одна общая переменная, то все вместе не имеет никакого смысла. Если, несмотря на это правило, мы попытаемся проинтерпретировать это выражение, получится несколько вариантов.
Выпишем все возможные побочные эффекты:

(x=1)&(y=1)
(x=2)&(y=2)

(x=2)&(y=1)

Каналы

Каналы позволяют обмениваться информацией и синхронизироваться в параллельных процессах.
Синтаксис:

channel
ch_def1,
…
ch_defN

где каждое из выражений ch_defi имеет вид:

id1, … , idk: type_expr

type_expr – это тип значений, которые можно передавать через этот канал.
Есть два типа операций: послать сообщение в канал и получить сообщение из канала. Семантика хранения и упорядочивания сообщений в канале не определяется.
Ограничения:

1) из канала нельзя прочитать сообщение до того (по времени), как соответствующая запись была сделана.
2) из канала нельзя дважды прочесть одно и тоже.

Описание функций:

f: type_expr1 -> access_def type_expr2

access_def – это объекты побочного эффекта.
Если речь идет о доступе к переменным, используются два ключевых слова:

read x1, y
write x2, z

Если переменные стоят после read, то в этой функции они будут доступны только на чтение, после write – на чтение и запись. Можно разрешить функции доступ ко всем переменным:

read all или **write** all

Рассмотрим следующую ситуацию:

f: … -> **write** x
/* в теле функции f используется функция g */
g: … -> **write** y

Это ошибка, т.к. в f сказано, что никакого побочного эффекта, кроме изменения x, нет.

Мы рассмотрели случай, когда функция имеет доступ только к переменным. Вторая разновидность доступа – это доступ к каналам. Для каналов вводятся ключевые слова in и out.
Пример:

channel c1: **Bool** , c2: **Int**
value f: **Unit** -> **in** c2 **out** c1 **Unit**

Есть два канала: для передачи булевских и целых данных. По своей сути эта функция является классификатором.

Запись в канал:

c2! <value_expr>

Тип <value_expr> совпадает с типом c2 (в данном примере <value_expr> = Int), тип всего выражения – Unit.

Аналогично производится чтение из канала:

c1?

В данном случае тип всего выражения – Bool.

Пример использования:

if c1? **then** …

Если произошло 2 записи в канал: c1!x и c1!y, значением c1? будет либо значение x, либо y. Должно быть также выполнено условие, что чтение происходит после записи.

Самостоятельная работа.

(y:=1)||(y:=2)

Выписать, следуя правилам RSL, результат выражения.

Ответ: y := (1 П 2), т.е. y наверняка получит значение 1 или 2.

П (или |^|)– **комбинатор внутреннего выбора**. Внутренний выбор – это ситуация, когда нет фактов, которые могут повлиять на результат.

□ (или |=|) – **комбинатор внешнего выбора**. Внешний выбор – результат зависит от некоторых причин. В качестве ответа мы говорим, что выполнено одно из двух выражений.

Синтаксис внешнего выбора:

value_expr1
□
value_expr2
□
...
□
value_exprN

В случае внешнего выбора сначала решаем, какая из ветвей будет работать, а потом проводим вычисления по выбранной ветви. Выбор происходит в зависимости от среды, т.е. от готовности окружения обмениваться сообщениями с этой веткой. Если в окружении команда c1!, то будет искаться ветвь с действием x:=c1?. Если в окружении y:=c2?, то, если она есть, будет выбрана ветка, например, с таким в выражением: c2!100.

Рассмотрим такой пример: (c1! || y:=c2?) || (c2!100 □ x:=c1?)

В этой ситуации может быть выбрана любая из двух ветвей внешнего выбора. Если пошли по ветке c2!100, то вычисляется y:=c2?, а процесс c1! остановится. В результате вычислений останется один

процесс, который готов взаимодействовать с окружением для него. Если есть несколько вариантов, мы должны расписать их через знак П.

Рассмотрим такое выражение:
e1|| (e2 || e3) || e4,
где e1, e2, e3, e4 – это некоторые выражения.
В дополнение к этому выражению описан список каналов.
Эта модель будет вести себя по-разному в зависимости от внутренней коммуникации между объявленными компонентами и коммуникациями снаружи.
Для упрощения задачи попробуем разобрать конкретные случаи поведения этого выражения. Как сузить границы рассмотрения для каждого конкретного случая? Один из способов – ввести приоритеты взаимодействий, но это сложно.
В RAISE использован другой подход. Введен оператор **interlock** (++ или ╫).
(e1|| (e2 || e3) || e4) ++ (e5)
Это означает, что сначала реализуются все взаимодействия выражения с компонентой e5, моделирующей условия внешней среды, и только потом рассматриваются внутренние взаимодействия. В этой схеме всего два уровня приоритетов.

Рассмотрим пример.
Пусть по всем каналам передаются целые значения.
((a!1;x:=b?) || (y:=b?) || (b!2; x:=b?)) ++ (b! a?)

Проведем упрощения:

1) надо получить информацию по каналу a
2) затем записать в канал b

После первого вычисления:
((x:=b?) || (y:=b?) || (b!2; x:=b?)) ++ (b!1)
Далее возможно три варианта, куда будет прочитано значение из канала b:
((x:=1) || (y:=b?) || (b!2; x:=b?)) П
((x:=b?) || (y:=1) || (b!2; x:=b?)) ~~П~~
~~(((x:=b?) || (y:=b?) || (b!2, x:=b?)) ++ (b!1))~~ /* единица должна оказаться на месте третьего b?, но это вычисление никогда не закончится по причине внутренней блокировки */
(((x:=b?) || (y:=b?) || (b!2; x:=b?)) ++ (b!1)) ≡ **stop**

Каждый элемент, записанный в канал, может быть прочитан только один раз.
Если при упрощении выражения мы пришли к ситуации e1 П e2 П ... П eК, где eК – это deadlock (т.е. нет возможности завершить вычисления), то говорим, что результатом значения может быть одно из получившихся выражений или deadlock. В данном примере мы пришли не к этой ситуации, потому что возможность завершить вычисления здесь есть. Поэтому третий вариант игнорируем, а не оставляем deadlock.
Продолжим рассматривать, как эти две возможности могут разрешиться.

У разделителя “;” есть функция упорядочивания во времени. Модель взаимодействия по каналам в RSL синхронная. Стороны могут начать взаимодействие в любом порядке, но завершатся оба процесса только после того, как оба объявят о своей готовности, т.е. завершаются они строго одновременно. Процесс обмена состоит из двух частей: передачи данных и синхронизации обоих процессов в данной точке. Поэтому в каждом случае только один вариант, кто прочтет двойку. Поэтому приходим к следующему упрощению:
((x:=1) || (y:=2) || (x:=b?)) П ((x:=2) || (y:=1) || (x:=b?))
Выражение ((x:=1) || (y:=2) || (x:=b?)) эквивалентно (x:=1; y:=2; x:=b?), поэтому итоговое выражение выглядит следующим образом:
(x:=1; y:=2; x:=b?) П (x:=2; y:=1; x:=b?)

По этой теме на коллоквиуме будет одна задача. Может быть два варианта:

- Упростить выражение с ++
- Упростить выражение с внутренними и внешними выборами в контексте if и case.