ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

УДК 681.324

С. А. Зинкин

СЕТИ АБСТРАКТНЫХ МАШИН ВЫСШИХ ПОРЯДКОВ В ПРОЕКТИРОВАНИИ СИСТЕМ И СЕТЕЙ ХРАНЕНИЯ И ОБРАБОТКИ ДАННЫХ (МЕХАНИЗМЫ ИНТЕРПРЕТАЦИИ И ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ)

Предложен логико-алгебраический подход к определению операционной семантики распределенных систем хранения и обработки данных, основанный на описании данных систем сетями абстрактных машин, при определении которых используется логика предикатов высших порядков. Предлагаемые модели и методы служат цели создания новой объектно-ориентированной сетевой технологии построения распределенных систем хранения и обработки данных на основе согласованных взаимодействий объектов через общее пространство – коммуникационную среду или общее пространство информационных объектов, а также сетевых, в том числе метакомпьютерных приложений для систем хранения и обработки данных на основе непосредственной интерпретации формальных спецификаций.

Введение

Общая методология проектирования, предлагаемая в настоящей работе, основана на использовании новых парадигм сетевых информационных технологий и моделей согласования и координации объектов и процессов через общее пространство функций и предикатов. В процессе проектирования используются логико-алгебраические модели систем распределенной и параллельной обработки информации. В разработанных моделях используется логический язык многосортного исчисления предикатов первой и высших ступеней, представляющий декларативные знания о предметной области. Сигнатура при этом должна задавать структурные связи между понятиями предметной области, представленные предикатами и функциями. Логические связи должны задаваться формулами, которые записываются в сигнатуре. Необходимо также формально представить знания о процессах хранения и обработки структурированных данных в распределенных системах. В отличие от обычных логических моделей искусственного интеллекта, используется представление процедурных знаний на основе аппарата алгебр операторов и условий с расширенной темпоральными операциями сигнатурой операций.

Разработанные формальные методы предлагается использовать в качестве основы для новой объектно-ориентированной сетевой технологии построения распределенных систем хранения и обработки данных на основе согласованных взаимодействий объектов; в том числе на этой основе воз-

можно создание сетевой операционной среды, адекватной решаемой задаче, т.е. сетевому приложению.

Ранее в работе [1] автором предложен базовый формализм – сети абстрактных машин высших порядков, берущих свое начало от машин Колмогорова–Успенского–Шёнхаге и Гуревича. При определении формализма использованы некоторые элементы алгебры алгоритмов Глушкова, отмеченные в [1]. Рассмотрим механизмы интерпретации данных машин с целью их использования при проектировании систем и сетей хранения и обработки данных.

Механизмы интерпретации сетей абстрактных машин

При использовании в правилах вывода операторов $\tilde{\forall}$ и $\tilde{\forall}!!$ возникает проблема согласования элементарных обновлений интерпретации сигнатуры $\sum$. Рассмотрим следующее правило вывода:

$$\frac{(\tilde{\forall} x \in X,\, y \in Y) q(x,\, y),\, (\tilde{\forall} z \in Z,\, t \in T) r(z,\, t)}{f\left(x,\, y,\, z\right) \xleftarrow{*} t}. \qquad (1)$$

Пусть, например, в результате выполнения квантифицированных операторов выбора, или вычисления термов, получены два отношения:

$$\{< x_1,\, y_2 >, < x_3,\, y_4 >\} \text{ и } \{< z_2,\ \ t_3 >,\ < z_2,\ \ t_4 >\}.$$

После выполнения операции декартова произведения будет сформировано следующее отношение:

$$\{< x_1,\, y_2,\, z_2, t_3 >, < x_1,\, y_2,\, z_2, t_4 >, < x_3,\, y_4,\, z_2, t_3 >, < x_3,\, y_4,\, z_2, t_4 >\},$$

а затем выполнятся 4 обновления тернарной функции *f*:

$$f(x_1,\, y_2,\, z_2) \leftarrow t_3,$$
$$f(x_1,\, y_2,\, z_2) \leftarrow t_4,$$
$$f(x_3,\, y_4,\, z_2) \leftarrow t_3,$$
$$f(x_3,\, y_4,\, z_2) \leftarrow t_4.$$

Последние обновления соответствуют случаю, когда в модуле CeAM или PCeAM при реализации правила (1) должен быть выполнен следующий блок:

$$B = \{f(x_1,\, y_2,\, z_2) \leftarrow t_3,\, f(x_1,\, y_2,\, z_2) \leftarrow t_4,$$
$$f(x_3,\, y_4,\, z_2) \leftarrow t_3,\, f(x_3,\, y_4,\, z_2) \leftarrow t_4\},$$

где запятая, разделяющая элементарные обновления функции, соответствует символу темпоральной операции «|» (выполнить «возможно одновременно», в произвольно выбранном порядке). Далее необходимо реализовать конкретные механизмы выполнения элементарных обновлений в блоке *B* некоторого модуля CeAM или PCeAM.

Механизм реализации предполагает интерпретацию модуля (узла CeAM или PCeAM) специальным агентом-сервером. Этот агент вначале вычисляет α-выражения (условия), выполняя при этом проверку значений атомарных формул и реализуя квантифицированные операторы выбора. К моменту выполнения некоторого блока значения всех предметных переменных, используемых в записях выражений данного блока, должны быть вычислены. Затем

агент выполняет модификацию (обновление) информационных объектов, представляющих функции и предикаты в некотором FS-пространстве (FS – Function Spaces). FS-пространство – это абстрактная структура данных, определенная функционально, посредством выполняемых над ней операций. Элементы FS-технологии подробно описаны в работах [2–5]. Особенностью предлагаемой технологии (FS-технологии) является то, что при необходимости несколько агентов могут интерпретировать один и тот же модуль, разделяя его локальную сигнатуру $\sum_m \subset \sum$, проверяя в параллельно-конвейерном режиме α-выражения и в конечном итоге выполняя обновления функций и предикатов в блоках. Продолжим рассмотрение выполнения правила (1). Каждый информационный объект в один и тот же момент времени может модифицироваться только одним агентом, поэтому рассматриваемый нами блок *B* реализуется одним агентом, который в произвольном порядке выполняет элементарные обновления блока. Так, если после выполнения обновления $f(x_1, y_2, z_2) \leftarrow t_3$ агент выполнит обновление $f(x_1, y_2, z_2) \leftarrow t_4$, то после выхода агента из блока *B* сохранится, естественно, результат последнего обновления, а результат предыдущего обновления аннулируется. Таким образом, поскольку в явной форме не был указан порядок выполнения обновлений в блоке *B*, результаты работы блока и, следовательно, модуля в целом даже при одинаковых исходных условиях могут различаться. Поэтому необходимо заранее провести анализ, устраивает ли разработчика распределенного приложения результат работы модуля.

Согласованными, например, следует считать обновления различных функций, которые могут быть реализованы различными агентами. Например, блок *Q* элементарных согласованных обновлений трех различных унарных функций

$$Q = \{q_1(x_1) \leftarrow y_1, q_2(x_1) \leftarrow y_2, q_3(x_2) \leftarrow y_4\}$$

может быть реализован тремя агентами (например, исходным агентом и двумя его копиями), причем временные интервалы работы данных агентов могут пересекаться в случае, если для исполнения агентов могут быть выделены три отдельных (физических) вычислительных узла.

Рассмотрим еще один пример выполнения согласованных обновлений функций и правил вывода. Пусть, например, некоторый запрос a_1 к распределенной реплицированной базе данных требует для своего выполнения ресурсов всех четырех серверов баз данных. На каждом сервере размещена соответствующая копия базы данных. Далее допустим, что для доступа к каждому серверу необходимо иметь право доступа, или так называемый «жетон». Для учета занятости жетонов используется унарный предикат $p: T \to \{\text{true}, \text{false}\}$, где T – множество жетонов. Для фиксации того факта, что для запросов выделены ресурсы, будем использовать бинарный предикат $q: A \times T \to \{\text{true}, \text{false}\}$, где A – множество запросов. Введем также унарный предикат $g: A \to \{\text{true}, \text{false}\}$, определяющий готовность запроса к исполнению. Запишем выражение для модуля *m*, реализующего выделение четырех единиц ресурса для выполнения запроса:

$$m = [(g(a_1)][([\tilde{\forall}!!t \in T)p(t)](\{q(a_1, \ t) \overset{*}{\longleftarrow} \text{true}, p(t) \overset{*}{\longleftarrow} \text{false}, \\ g(a_1) \leftarrow \text{false}\} \vee R^E) \vee R^E). \qquad (2)$$

В процессе выполнения данного модуля в соответствии с выражением (2) осуществляется проверка наличия запроса a_1 (проверяется, истинно ли высказывание $g(a_1)$), затем вычисляется терм $(\tilde{\forall}!!t \in T)p(t)$. В случае, если истинно высказывание $(\forall t \in T)p(t)$, что означает доступность всех единиц ресурса, далее выполняются правила обновления предикатов:

$$q(a_1, t) \overset{*}{\longleftarrow} \text{true},\ p(t) \overset{*}{\longleftarrow} \text{true} \text{ и } g(a_1) \leftarrow \text{false}.$$

Оператор $\overset{*}{\longleftarrow}$ реализует необходимые обновления предикатов, причем в блоке модуля *m* выполняются следующие элементарные обновления:

$$\{q(a_1, t_1) \leftarrow \text{true}, q(a_1, t_2) \leftarrow \text{true}, q(a_1, t_3) \leftarrow \text{true},$$
$$q(a_1, t_4) \leftarrow \text{true}, p(t_1) \leftarrow \text{false}, p(t_2) \leftarrow \text{false}, p(t_3) \leftarrow \text{false},$$
$$p(t_4) \leftarrow \text{false}, g(a_1) \leftarrow \text{false}\}.$$

К терму $(\tilde{\forall}!!t \in T)p(t)$ применен оператор «подчеркивание» по умолчанию. Напомним, что данный оператор ставит в соответствие оператору $\tilde{\forall}!!$ высказывание, истинное в том случае, если выполнение данного оператора успешно завершено. Символ «_» подчеркивания в α-выражениях модулей мы будем опускать, если α-выражение является единственным выражением в квадратных скобках; будем также опускать и звездочку в операторе $\overset{*}{\longleftarrow}$ в тех случаях, когда это не вызывает недоразумений.

В приведенном выше примере локальную сигнатуру модуля *m* образуют только предикаты. Заменим теперь предикат *q* функцией $f : T \to A$ и запишем новое выражение для модуля

$$m' = [g(a_1)]\left(\left[(\tilde{\forall}!!t \in T)p(t)\right]\right.$$
$$\left.\left(\{f(t) \overset{*}{\longleftarrow} a_1,\ p(\text{t}) \overset{*}{\longleftarrow} \text{false},\ g(a_1) \leftarrow \text{false}\} \vee R^E\right) \vee R^E\right). \qquad (3)$$

При условии истинности обоих проверяемых α-выражений в блоке модуля m' выполняются следующие элементарные обновления:

$$\{f(t_1) \leftarrow a_1, f(t_2) \leftarrow a_1, f(t_3) \leftarrow a_1, f(t_4) \leftarrow a_1, p(t_1) \leftarrow \text{false},$$
$$p(t_2) \leftarrow \text{false}, p(t_3) \leftarrow \text{false}, p(t_4) \leftarrow \text{false}, g(a_1) \leftarrow \text{false}\}.$$

Теперь уже функция *f* связывает каждый жетон с запросом к распределенной базе данных. Модифицируем поставленную выше задачу. В более общей постановке задачи клиенты распределенной базы данных формируют множество запросов. Модуль, описываемый представленным ниже выражением, реализует выбор запроса и «связывание» его с жетонами для последующего выполнения:

$$m'' = \left[(\tilde{\exists}!x \in A)g(x)\right]\left(\left[(\tilde{\forall}!!t \in T)p(t)\right]\right.$$
$$\left.\left(\{q(x, t) \overset{*}{\longleftarrow} \text{true},\ p(t) \overset{*}{\longleftarrow} \text{false},\ g(x) \leftarrow \text{false}\} \vee R^E\right) \vee R^E\right). \qquad (4)$$

Другой вариант записи модуля основан на использовании функции f для связывания значений предметных переменных x и t:

$$m''' = \left[(\tilde{\exists}!x \in A) g(x) \right] \left(\left[(\tilde{\forall}!!t \in T) p(t) \right] \right.$$
$$\left. \left(\{ f(t) \xleftarrow{*} x,\ p(t) \xleftarrow{*} \text{false},\ g(x) \leftarrow \text{false} \} \vee R^E \right) \vee R^E \right). \qquad (5)$$

Заметим, что α-выражения в каждом из выражений (2)–(5) не зависят друг от друга, поэтому порядок проверки условий в модулях можно изменять и данные модули могут быть описаны следующими эквивалентными выражениями:

$$m = \left[(g(a_1) \,\&\, \underline{(\tilde{\forall}!!t \in T) p(t)} \right]$$
$$\left(\{ q(a_1, t) \xleftarrow{*} \text{true},\ p(t) \xleftarrow{*} \text{false},\ g(a_1) \leftarrow \text{false} \} \vee R^E \right); \qquad (6)$$

$$m' = \left[g(a_1) \,\&\, \underline{(\tilde{\forall}!!t \in T) p(t)} \right]$$
$$\left(\{ f(\ t) \xleftarrow{*} a_1,\ p(t) \xleftarrow{*} \text{false},\ g(a_1) \leftarrow \text{false} \} \vee R^E \right); \qquad (7)$$

$$m'' = \left[\underline{(\tilde{\exists}!x \in A) g(x)} \,\&\, \underline{(\tilde{\forall}!!t \in T) p(t)} \right]$$
$$\left(\{ q(x, t) \xleftarrow{*} \text{true},\ p(t) \xleftarrow{*} \text{false},\ g(x) \leftarrow \text{false} \} \vee R^E \right); \qquad (8)$$

$$m''' = \left[\underline{(\tilde{\exists}!x \in A) g(x)} \,\&\, \underline{(\tilde{\forall}!!t \in T) p(t)} \right]$$
$$\left(\{ f(t) \xleftarrow{*} x,\ p(t) \xleftarrow{*} \text{false},\ g(x) \leftarrow \text{false} \} \vee R^E \right). \qquad (9)$$

При физической реализации модулей (6)–(9) время выполнения модуля в реальной сетевой среде может в существенной степени зависеть от порядка проверки условий.

Сети СеАМ функционируют, переходя от одной интерпретации сигнатуры $I(t_i)$ к другой – $I(t_j)$, $(t_j > t_i)$, начиная с какой-либо начальной интерпретации сигнатуры $I(t_0)$. Смена интерпретации соответствует модификации пространства функций и предикатов, или FS-пространства. Для инициирования сети должна быть выполнена серия обновлений некоторой стартовой функции f_{start}. Эти обновления выполняет стартовый агент a_s. Например, после выполнения серии обновлений $f_{\text{start}}(m_1) \leftarrow a_1$, $f_{\text{start}}(m_2) \leftarrow a_1$ и $f_{\text{start}}(m_3) \leftarrow a_2$ агент-сервер a_1 сможет поочередно интерпретировать модули m_1 и m_2, а агент-сервер a_2 сможет интерпретировать модуль m_3. Предполагается, что до начала стартового агента $f_{\text{start}}(m_1)$ = undef, $f_{\text{start}}(m_2)$ = undef_1 и $f_{\text{start}}(m_3)$ = undef, где undef – неопределенная константа. После выполнения указанных обновлений агенты-серверы a_1 и a_2 будут интерпретировать специальным образом закодированные в виде команд некоторой виртуальной машины выражения для модулей, воспринимая эти команды как сценарии своей работы. В ряде случаев для повышения эффективности управления сетью абстрактных машин целесообразно осуществить декомпозицию функции f_{start}, разбив ее (горизонтально) на функции, число которых совпадает с числом модулей сети: f_{start1}, f_{start2} и f_{start3}. Если теперь разместить информационные объекты, соответст-

вующие этим функциям, на различных физических узлах, то сеть из трех модулей начнет функционировать после того, как будут обновлены три различные стартовые функции: $f_{\text{start1}}(m_1) \leftarrow a_1$, $f_{\text{start2}}(m_2) \leftarrow a_1$ и $f_{\text{start3}}(m_3) \leftarrow a_2$. Пусть модули СеАМ m_1, m_2 и m_3 будут размещены на физических узлах y_1, y_2 и y_3 соответственно. Тогда агент-сервер a_1 будет интерпретировать модули m_1 и m_2, поочередно перемещаясь между узлами y_1 и y_2, а агент-сервер a_3 будет постоянно находиться на узле y_3 и интерпретировать модуль m_3. Таким образом, агент-сервер a_1 должен обладать свойством мобильности. Возможно, такая организация управления работой сети окажется неэффективной, тогда целесообразнее организовать работу стационарных агентов-серверов, размещенных на тех же узлах, на которых размещены соответствующие модули.

Определение 1. Модули сетей СеАМ или РСеАМ эквивалентны, если они одинаковым образом модифицируют текущую интерпретацию одной и той же локальной сигнатуры Σ_m.

Последнее определение проиллюстрируем некоторыми примерами. Из представленного выше описания механизмов интерпретации следует, что для спецификации операционной семантики реализаций сетей абстрактных машин базового формализма из работы [1] недостаточно. Для полноты спецификации необходимо рассматривать и работу агентов-серверов, интерпретирующих выражения, которыми описываются модули СеАМ и РСеАМ. В этой связи рассмотрим работу некоторых модулей СеАМ и РСеАМ. Данные примеры будут также полезны для эквивалентных преобразований выражений для модулей СеАМ и РСеАМ.

Результат работы модуля СеАМ III рода

$$m_1/a_1 = [\underline{(\tilde{\exists}!x \in X)p(x)}]\Big(\{B;\ p(x) \leftarrow \text{false}\} \vee R^E\Big)\Big/a_1,$$

где дробной чертой отделено имя агента-сервера a_1, интерпретирующего данное выражение, будет таким же, как и результат работы модуля того же рода:

$$m_2/a_2 = [\underline{(\tilde{\forall}x \in X)p(x)}]\Big(\{B;\ p(x) \xleftarrow{*} \text{false}\} \vee R^E\Big)\Big/a_2.$$

В обоих случаях предполагается, что в выражении B не выполняются действия, приводящие к изменению значений α-условий. Символы операции подчеркивания в обоих случаях могут быть опущены, т.к. подчеркнутые выражения – единственные в квадратных скобках.

Работа обоих модулей предполагает предварительную блокировку всех функций и предикатов, имена которых составляют одну и ту же локальную сигнатуру Σ_m. Под блокировкой функций и предикатов подразумевается, естественно, блокировка соответствующих информационных объектов, представляющих данные функции и предикаты в физической реализации FS-пространства. При работе модуля $m_1/a1$ происходит циклическое выполнение блока $\{B;\ p(x) \leftarrow \text{false}\}$ до тех пор, пока не станет пустой область истинности предиката p. Агент-сервер a_1 при этом многократно выполняет данный блок; на время выполнения цикла предикаты и функции с именами из локальной сигнатуры Σ_m остаются заблокированными для других модулей.

Модуль m_2/a_2, работа которого завершается тем же результатом, выполняется иначе. Проверив α-условие $\underline{(\tilde{\forall}x \in X)p(x)}$, агент-сервер a_2 одно-

временно «выберет» все значения предметной переменной x из области истинности предиката p, а затем выполнит множественные правила обновления в блоке $\{B;\ p(x)\overset{*}{\longleftarrow}\text{false}\}$. Если агент-сервер a_2 выберет не менее одного значения переменной x, то данный блок выполнится один раз. После выполнения множественного обновления $p(x)\overset{*}{\longleftarrow}\text{false}$ логическое условие станет ложным, т.к. в области истинности предиката p не останется ни одного элемента, и работа модуля m_2/a_2 завершится.

Двум описанным выше СеАМ-модулям эквивалентен следующий РСеАМ-модуль:

$$m_3/a_3 = \left[\underline{\neg\left((\tilde{\exists}!x\in X)p(x)\right)}\right]\{B;\ p(x)\leftarrow\text{false}\}\Big/a_3,$$

построенный на основе операции α-итерации. Здесь блок $\{B;\ p(x) \leftarrow \text{false}\}$ выполняется циклически до тех пор, пока ложно условие $\underline{\neg\left((\tilde{\exists}!x\in X)p(x)\right)}$.

На основании приведенного выше описания сформулируем следующее уточненное определение модуля СеАМ или РСеАМ, дополнив тем самым определение 6.

Определение 2. Модулем в реализации сетей СеАМ или РСеАМ называется пара $<m, a>$, где m – СеАМ- или РСеАМ-выражение, a – интерпретирующий это выражение агент-сервер. В случае, когда несколько агентов-серверов $a_1, a_2, \ldots, a_n$ интерпретируют одно и то же выражение m в параллельно-конвейерном режиме, модулем реализации сетей СеАМ или РСеАМ будем называть $(n+1)$-арный кортеж $< m, a_1, a_2, \ldots, a_n >$, а кортеж $< a_1, a_2, \ldots, a_n >$ назовем мультиагентом.

В дальнейшем чаще наличие агента-сервера, если это не противоречит контексту, будем только подразумевать, отождествляя для простоты модуль с интерпретируемым агентом выражением.

При анализе дискретных сетевых моделей широкого класса, как правило, полагается, что каждому состоянию FS-пространства соответствует вершина графа достижимых состояний (ГДС). Модификация текущей интерпретации I сигнатуры Σ приводит к смене состояния. Графом достижимых состояний сетевой модели N (следуя, например, работе [6]) назовем ориентированный граф $G = (X, U, R)$, где X – множество вершин, соответствующее (взаимно-однозначно) множеству достижимых состояний сетевой модели; $U \subseteq X\times X$ – множество дуг такое, что $(x_i, x_j) \in U$, если состояние x_j непосредственно достижимо из состояния x_i; R: $U\rightarrow M$ – функция разметки дуг именами модулей такая, что $R(x_i, x_j) = m_k$, если состояние x_j непосредственно достижимо из состояния x_i при выполнении модуля m_k. Далее при необходимости производится автоматический или визуальный анализ свойств сетевой модели по графу G. Основой верификации сетевых моделей систем хранения и обработки данных может быть временная логика, в частности логика дерева вычислений (Computation Tree Logic, или, сокращенно, CTL). Вопросы верификации на базе временной логики достаточно полно представлены в отечественной и зарубежной литературе. Соответствующие методы и программные средства подробно описаны, например, в руководстве [6].

Декомпозиция сетей абстрактных машин

Сложные модули СеАМ удобно в ряде случаев представлять в виде совокупности простых модулей. Подобные представления часто используются при переходе к записям алгоритмов в виде систем продукций. Например, модуль

$$m_0 = [\alpha_1]([\alpha_2](B_1 \vee B_2) \vee [\alpha_3](B_3 \vee [\alpha_4](B_4 \vee B_5))) \quad (10)$$

представим в виде совокупности $M = \{m_1, m_2, m_3, m_4, m_5\}$ модулей:

$$m_1 = [\alpha_1 \& \alpha_2](B_1 \vee R^E), \quad (11)$$

$$m_2 = [\alpha_1 \& \neg\alpha_2](B_2 \vee R^E), \quad (12)$$

$$m_3 = [\neg\alpha_1 \& \alpha_3](B_3 \vee R^E), \quad (13)$$

$$m_4 = [\neg\alpha_1 \& \neg\alpha_3 \& \alpha_4](B_4 \vee R^E), \quad (14)$$

$$m_5 = [\neg\alpha_1 \& \neg\alpha_3 \& \neg\alpha_4](B_5 \vee R^E). \quad (15)$$

В реальной сетевой среде, при наличии множества агентов-серверов, интерпретирующих многочисленные модули СеАМ, результаты работы исходного модуля и совокупности модулей могут различаться, т.к. различные, может быть и не входящие в совокупность M модули (11)–(15), могут изменять значения логических условий в различные моменты времени.

Подобное преобразование модуля СеАМ в общем случае не является эквивалентным, т.к. для интерпретации вновь полученных модулей, возможно, организуется новая распределенная операционная среда – физические узлы и агенты-серверы; в результате новые модули, возможно, будут иначе влиять на логику функционирования сети. Поясним сказанное на следующем примере. Представим модуль

$$m_1 = [\alpha](r_1, r_2 \vee R^E)$$

в виде совокупности двух модулей

$$m_{11} = [\alpha](r_1 \vee R^E)$$

и

$$m_{12} = [\alpha](r_2 \vee R^E),$$

которые могут выполняться «возможно одновременно», т.к. это было ранее предписано операцией «,» применяемой в блоке модуля m_1 (символ запятой в данном случае используется для простоты записи вместо символа темпоральной операции «$|$» – «выполнить возможно одновременно»). Однако режим выполнения модулей m_{11} и m_{12} может измениться. При реализации этих модулей в распределенной среде они конкурируют за право первоочередного вычисления и последующей проверки условия α, которое выступает в данном случае в роли неразделяемого ресурса. Выполняющееся затем правило обновления одного из модулей может так повлиять на условие α, что оно окажется ложным непосредственно до выполнения другого модуля. Кроме того,

условие α может стать ложным и в результате работы какого-либо третьего модуля. В результате одно из элементарных правил обновления не будет выполнено. Подобной проблемы не возникает для исходного модуля m_1, поскольку к началу его выполнения специальный агент-сервер блокирует функции и предикаты с символами из локальной сигнатуры Σ_{m1} данного модуля, и далее ничто не препятствует выполнению заранее согласованных обновлений r_1 и r_2. Это означает, что представление исходного модуля m_1 в виде двух взаимодействующих модулей в общем случае нельзя назвать эквивалентным. Похожее явление наблюдается, например, и при декомпозиции модуля

$$m_2 = [\alpha]\left(\{r_1; r_2\} \vee R^E\right),$$

и представлении его совокупностью двух выполняемых последовательно модулей

$$m_{21} = [\alpha](r_1 \vee R^E)$$

и

$$m_{22} = [\alpha](r_2 \vee R^E),$$

т.е. когда СеАМ-модуль m_2 заменяется составным РСеАМ-модулем

$$m_2' = (m_{21}; m_{22}) = (m_{21} \uparrow m_{22}),$$

где символ темпоральной операции «$\uparrow$» для простоты может быть заменен символом «точка с запятой». Здесь возможна ситуация, когда после выполнения элементарного обновления r_1 условие α станет ложным и модуль m_{12} не сможет выполниться. В распределенной среде его выполнение возможно спустя какое-то время, например, после того, как некоторый другой модуль изменит значение логического условия α на истинное.

Данная ситуация характерна и для многих других классов распределенных вычислительных систем, в которых реализуется система взаимодействующих алгоритмических модулей. Результат проведенной декомпозиции должен быть тщательно проанализирован разработчиком распределенной системы на предмет ее приемлемости и целесообразности с точки зрения правильного и эффективного решения системой общей задачи. Полученная в результате декомпозиции система может оказаться не эквивалентной исходной, но ее структура и результат функционирования могут удовлетворять потребителя.

Применение логики высших порядков к построению сетей абстрактных машин

Выше было отмечено, что в логиках высшего порядка допускается использование предикатных и функциональных переменных и кванторов по этим переменным. Здесь наряду с предметными константами и переменными определенных сортов (типов) используются функциональные и реляционные константы и переменные с определенной сигнатурой. Дополним приведенное ранее описание введенных нами СеАМ высших порядков конкретными примерами.

В работе [7] указывается, что преимуществами логики второго порядка со стандартной семантикой являются ясность и соответствие интуиции, а ее выразительные возможности в качестве оснований математики намного богаче выразительных возможностей логики первого порядка. Не случайно в «Principia Mathematica» Рассел и Уайтхед в основания математики положили логику второго и высшего порядков, а Гильберт и Аккерман в работе «Основы математической логики» дали четкое разделение ролей языка первого порядка и языков высших порядков. В этой же работе отмечается, что значительное число математических концепций, например таких, как математическая индукция, упорядочение, конечность, кардинальность и др., не могут быть выражены на языке первого порядка, но все эти понятия выразимы на языке второго порядка. В другой работе [8] отмечается, что вся теоретико-множественная проблематика может быть сформулирована во второпорядковых терминах. В работе [9] рассматривается многосортная первопорядковая логика в качестве переинтерпретации второпорядковой логики и логики высших порядков, что позволяет сделать вывод о перспективности использования многосортных логик различных порядков в качестве основы языка для определения абстрактных машин.

В выражениях для модулей CeAM и PCeAM (модуль в данном контексте можно назвать «локальным модификатором» интерпретации сигнатуры многоосновной алгебраической системы) важную роль играют предикаты сравнения, в том числе предикат равенства. Рассмотрим в качестве примеров выражения для модулей CeAM и PCeAM второго порядка, поясняющие специфику реализации квантифицированных операторов выбора $\tilde{\exists}!$ и $\tilde{\forall}$. Модуль CeAM e_1 заменяет значения функций на неопределенные в случае равенства их значений:

$$e_1 = \Big[q(a_0) \& \underline{(\tilde{\forall} x \in X,\, y \in Y)\big(p_1(x) \& p_2(y)\big)} \Big]$$
$$\Big(\Big[(\tilde{\forall} \varphi \in \Phi,\, \psi \in \Psi)\big(\varphi(x) = \psi(y)\big) \Big]$$
$$\Big(\{ \varphi(x) \xleftarrow{*} \text{undef},\, \psi(y) \xleftarrow{*} \text{undef},\, q(a_0) \leftarrow \text{false} \} \vee R^E \Big) \vee R^E \Big).$$

Здесь унарные предикаты p_1 и p_2 выполняют роль характеристических функций множеств X и Y, по которым пробегают предметные переменные x и y соответственно, а функциональные переменные φ и ψ пробегают соответственно по множествам функций Φ и Ψ. В процессе выполнения модуля e_1 вначале первый оператор $\tilde{\forall}$ формирует декартово произведение множеств X и Y. Затем второй оператор $\tilde{\forall}$ выбирает все такие пары значений функциональных переменных φ и ψ, для которых $\varphi(x) = \psi(y)$. Далее в единственном блоке модуля e_1 (выражение в фигурных скобках) выполняются множественные обновления выбранных функций. Унарный предикат q используется для того, чтобы блок модуля e_1 выполнился один раз. Оператор «_» («подчеркивание») ставит в соответствие выражению над чертой булеву переменную, принимающую значение true в случае, если оператор выполнен успешно.

Механизм пошаговой реализации квантифицированного оператора выбора $\tilde{\forall}$ скрыт. Раскроем действие этого механизма на примере двух следую-

щих модулей. Модуль e_2 выполняет такую же локальную модификацию интерпретации сигнатуры, что и модуль e_1:

$$e_2 = \Big[(\tilde{\exists}!x \in X,\ y \in Y)\big(p_1(x)\&p_2(y)\big)\Big]\Big[\big[(\tilde{\exists}!\varphi \in \Phi, \psi \in \Psi)\big(\varphi(x) = \psi(y)\big)\big]$$
$$\Big(\{\varphi(x) \leftarrow \text{undef}, \psi(y) \leftarrow \text{undef},\ p_1(x) \leftarrow \text{false},\ p_2(y) \leftarrow \text{false}\} \vee R^E\Big) \vee R^E\Big).$$

Для того чтобы выполнение модуля e_2 было эквивалентно выполнению модуля e_1, модуль e_2 должен выполняться многократно, пока не будет завершен перебор всех пар значений предметных переменных x и y. Многократное выполнение модуля e_2 обеспечивается некоторым агентом-сервером, который «рассматривает» выражение для данного модуля как описание сценария своих действий. Выполняя первый из операторов $\tilde{\exists}!$, агент-сервер выбирает очередную пару значений предметных переменных x и y. При этом он проверяет истинность конъюнкции $p_1(x)\&p_2(y)$. Далее при выполнении второго оператора $\tilde{\exists}!$ агент-сервер выбирает одну пару значений функциональных переменных φ и ψ, такую, что $\varphi(x) = \psi(y)$. В случае успешного выполнения обоих операторов выбора агент-сервер выполнит по одному обновлению функций, имена которых являются выбранными значениями функциональных переменных φ и ψ, а также по одному обновлению унарных предикатов p_1 и p_2 (характеристических функций множеств X и Y соответственно). В результате двух последних обновлений выбранная ранее пара значений предметных переменных x и y исключается из дальнейшего рассмотрения, а агент-сервер снова переходит к проверке условий применимости квантифицированных операторов выбора. Неявно заданный подобным образом цикл повторяется до тех пор, пока не будет завершено рассмотрение последней пары значений предметных переменных x и y. Таким образом, обеспечивается совпадение результатов работы модулей e_1 и e_2.

В ряде случаев может оказаться удобным использование расширенного варианта формализма РСеАМ, в котором более явно представлена процедурная компонента представления знаний о предметной области. Например, функцию СеАМ-модуля e_2 может выполнить следующий РСеАМ-модуль:

$$\mathrm{e}_3 = \Big[\underline{\neg(\tilde{\exists}!x \in \bar{X},\ y \in Y)\big(p_1(x)\&p_2(y)\big)}\Big]\Big\{\Big[(\tilde{\exists}!\varphi \in \Phi, \psi \in \Psi)\big(\varphi(x) = \psi(y)\big)\Big]$$
$$(\varphi(x) \leftarrow \text{undef}, \psi(y) \leftarrow \text{undef} \vee R^E),\ p_1(x) \leftarrow \text{false},\ p_2(y) \leftarrow \text{false}\Big\}.$$

Здесь цикличность выполнения действий задана α-итерацией, причем в качестве α-условия используется следующее выражение:

$$\underline{\neg(\tilde{\exists}!x \in X,\ y \in Y)\big(p_1(x)\&p_2(y)\big)}.$$

Выражения, которыми описывается функционирование модулей абстрактных машин, удобны для преобразований и непосредственной интерпретации и могут быть использованы в качестве основы для реализации языка непосредственно интерпретируемых спецификаций.

Приведенные примеры показывают, что использование многосортных логик и логик высших порядков позволяет плодотворным образом сочетать как декларативные, так и процедурные подходы к построению логико-алгебраических моделей распределенных систем хранения и обработки данных.

Особенности выполнения правил обновления текущей интерпретации сигнатуры

Рассмотрим следующее правило вывода:

$$\frac{(\tilde{\exists}!x\in X)\,p(x),\,(\tilde{\exists}!\,y\in Y)q(y),\,(\tilde{\exists}!z\in Z)r(z)}{f((\tilde{\exists}!x\in X)\,p(x),\,(\tilde{\exists}!\,y\in Y)q(y))\leftarrow(\tilde{\exists}!z\in Z)r(z)}. \qquad (16)$$

Это правило содержит термы, построенные на основе квантифицированных операторов выбора. Вывод в данном случае реализуется до конца лишь в том случае, если все операторы $\tilde{\exists}!$ выполнились успешно. Если в выражении для модуля CeAM или PCeAM хотя бы один из используемых в правиле (16) термов принимает значение undef, то вывод не может быть завершен. В принятой интерпретации формализмов CeAM и PCeAM данный факт соответствует случаю, когда значением выражения, описывающего модуль, является тождественное (пустое) обновление R^E интерпретации сигнатуры $\sum$.

Допустим, что вычислены все термы в правиле (16):

$$(\tilde{\exists}!x\in X)\,p(x)=x',\,(\tilde{\exists}!\,y\in Y)q(y)=y',\,(\tilde{\exists}!z\in Z)r(z)=z',$$

тогда в блоке соответствующего модуля далее будет выполнено обновление функции *f*:

$$f(x',\,y')\leftarrow z'.$$

Аналогично выполняется вывод и в случае применения оператора $\tilde{\exists}!!$.

Рассмотрим далее случаи, когда в правилах вывода используются термы, построенные на основе операторов $\tilde{\forall}$ и $\tilde{\forall}!!$. Результатом вычисления подобного терма является не константа или кортеж, а отношение, выбираемое из области истинности соответствующего предиката.

Пусть при работе некоторого модуля CeAM или PCeAM должно быть реализовано следующее правило вывода:

$$\frac{(\tilde{\exists}!x\in X,\,y\in Y)q(x,\,y),\,(\tilde{\forall}z\in Z,t\in T)r(z,t)}{p(x,\,y,\,z,\,t)\xleftarrow{*}\text{true}}, \qquad (17)$$

где q, r – бинарные, а p – 4-арный предикатные символы. В данном случае результатом вычисления терма $(\tilde{\exists}!x\in X,\,y\in Y)p(x,\,y)$ должен быть кортеж (например, $<x',\,y'>$), а результатом вычисления терма $(\tilde{\forall}z\in Z,t\in T)r(z,t)$ – некоторое отношение, или множество кортежей (например $\{<z',t'>,\ <z'',t''>\}$). В процессе выполнения правила вывода (9) после вычисления термов реализуется операция декартова произведения результирующих отношений:

$$\{<x',\,y'>\}\times\{<z',t'>,<z'',t''>\}=\{<x',\,y',\,z',t'>,<x',\,y',\,z'',t''>\}.$$

Предварительно кортеж $<x',\,y'>$ – результат вычисления первого терма, преобразуется в одноэлементное множество с помощью функции $f_{\text{singleton}}(<x',\,y'>)=\{<x',\,y'>\}$.

Далее выполняется правило $p(x, y, z, t) \xleftarrow{*} \text{true}$ обновления 4-арного предиката p. Звездочка, поставленная на стрелке, означает, что должно быть выполнено несколько согласованных элементарных обновлений предиката p:

$$p(x', y', z', t') \leftarrow \text{true},$$
$$p(x', y', z'', t'') \leftarrow \text{true},$$

и на этом вывод завершается.

Выводы

1. Предложены методы интерпретации сетей абстрактных машин высших порядков со структурированной памятью. Данные сети при их применении в качестве базового формализма в существенной степени расширяют возможности общей методологии проектирования распределенных систем согласования и координации процессов и объектов, основанной на использовании новых парадигм сетевых информационных технологий.

2. Предлагаемые модели и методы могут быть использованы в качестве основы при создании новой объектно-ориентированной сетевой технологии проектирования распределенных систем хранения и обработки данных на основе согласованных взаимодействий объектов через общее пространство – коммуникационную среду или общее пространство информационных объектов.

3. Сети абстрактных машин высших порядков в качестве объектов непосредственной интерпретации особенно удобны при создании такого сетевого программного обеспечения реконфигурируемых систем хранения и обработки структурированных данных, в котором стираются грани между сетевой операционной системой, распределенной системой управления базой данных и распределенным приложением.

Список литературы

1. **Зинкин, С. А.** Сети абстрактных машин высших порядков в проектировании систем и сетей хранения и обработки данных (базовый формализм и его расширения) / С.А. Зинкин // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2007. – № 3. – С. 13–22.
2. **Zinkin, S. A.** Distributed evolving algebras meet Java aglets: executable specifications of virtual mobile metacomputing on the Internet platform / S. A. Zinkin // New Information Technologies and Systems: Proceedings of the Third International Conference of Science and Technology. – 1998. – P. 50–52.
3. **Зинкин, С. А.** Концептуальная схема реализации сетевой информационной технологии FUNCTION SPACES / С. А. Зинкин // Новые информационные технологии и системы : материалы VI Международной научно-технической конференции. – Пенза : Изд-во Пенз. гос. ун-та, 2004. – С. 208–217.
4. **Зинкин, С. А.** Алгебраические методы спецификации операционной семантики сетей хранения и обработки данных с изменяемой топологией / С. А. Зинкин // Актуальные проблемы науки и образования : труды Международного юбилейного симпозиума. – Пенза : Информационно-издательский центр ПГУ, 2003. – 2 т. – С. 409–418.
5. **Зинкин, С. А.** Интеграция сетевых и информационных технологий на основе парадигм искусственного интеллекта / С. А. Зинкин // Новые информационные технологии и системы : материалы VII Международной научно-технической конференции. – Пенза : Изд-во Пенз. гос. ун-та, 2006. – С. 108–117.

6. **Дубинин, В. Н.** Языки логического программирования в проектировании вычислительных систем и сетей / В. Н. Дубинин, С. А. Зинкин. – Пенза : Изд-во Пенз. гос. ун-та, 1997. – 88 с.
7. **Целищев, В. В.** Язык математики и цели математического дискурса / В. В. Целищев // Философия науки. – 2003. – № 1 (16). – С. 18–45.
8. **Карпенко, А. С.** Логика на рубеже тысячелетий / А. С. Карпенко // Logical Studies (online journal). – 2000. – № 5. – P. 1–50.
9. **Manzano, M.** Extentions of first order logic. Cambridge tracts in theoretical computer science / M. Manzano. – Cambridge : Cambridge University Press, 1996. – 410 p.