Модел. и анализ информ. систем. Т. **21**, № **6** (2014) 31–43

УДК 517.51+514.17

Верификация алгоритмов мультиагентного анализа данных с помощью системы проверки моделей SPIN

Гаранина Н.О., Бодин Е.В., Сидорова Е.А.[1]

Институт систем информатики им. А. П. Ершова СО РАН
630090 Россия, г. Новосибирск, проспект Академика Лаврентьева, 6

e-mail: {garanina,bodin,lena}@iis.nsk.su

получена 16 октября 2014

Ключевые слова: пополнение онтологий, мультиагентные системы, проверка моделей, система верификации SPIN

В статье представлен подход к формальной верификации алгоритмов мультиагентного анализа данных для пополнения онтологий. Агенты системы на основе входных данных устанавливают значения элементов объектов, полученных на предварительной стадии анализа. Агенты параллельно осуществляют проверку семантической и синтаксической согласованности, используя правила пополнения онтологий и обработки данных. Поскольку агенты действуют параллельно, необходимо верифицировать некоторые важные свойства системы, связанные с этим, например, свойство корректности определения завершения работы системы. В нашем подходе используется инструмент проверки моделей SPIN. Протоколы агентов записаны на языке Promela, а свойства мультиагентной системы анализа данных выражены в логике LTL. Мы провели ряд экспериментов по проверке данной модели.

1. Введение

Целью статьи является представление подхода к формальной верификации мультиагентных алгоритмов анализа данных для пополнения онтологий.

Пусть у нас есть онтология, элементами которой являются классы, определяемые набором атрибутов, и отношения, которые определяются парой классов и набором атрибутов. Правила пополнения онтологии позволяют сформировать объекты классов и отношений данной онтологии. Кроме того, имеются правила обработки входных данных. Эти данные могут быть как неструктурированными (естественный язык), так и полуструктурированными (различные базы данных или помеченные веб-страницы). Мы считаем, что все эти правила определены формально таким

[1]Работа выполнена при финансовой поддержке РФФИ (проект № 13-01-00643-а) и президиума РАН (интеграционный проект СО РАН № 15/10 "Математические и методологические аспекты интеллектуальных информационных систем").

образом, что каждое правило (1) использует в качестве входных данных либо экземпляры классов или отношений и значения их атрибутов, либо формальные признаки входных данных; (2) может связывать набор значений входных атрибутов в экземпляр какого-либо класса; (3) может определять значение атрибутов отношения и принадлежность экземпляров заданному отношению.

Мультиагентный анализ данных для пополнения онтологии является многоуровневым процессом. На первой стадии предварительный анализ данных порождает недоопределенные объекты, которые могут быть экземплярами классов или отношений заданной онтологии. На следующей стадии эти объекты доопределяются, насколько это возможно, на основе входных данных и проверки семантической и синтаксической согласованности, используя правила пополнения онтологий и обработки данных. На третьей стадии эти объекты-экземпляры должны разрешить различные неоднозначности в своих определениях, которые являются неотъемлемой частью автоматического анализа данных.

На второй стадии анализа возникают *информационные агенты (агенты-экземпляры и агенты-отношения)*. Они соответствуют экземплярам классов и отношений онтологии. Информационные агенты взаимодействуют с *агентами-правилами*, которые реализуют заданные правила обработки данных и пополнения онтологий. Агенты обмениваются информацией, необходимой для определения значений атрибутов и новых связей/объектов информационных агентов. Специальный *агент-контролер* определяет завершение работы системы, т.е. момент, когда вся возможная информация извлечена из данных и все агенты находятся в режиме ожидания сообщений. В отличие от других агентов модели, этот служебный агент универсален, т.е. не зависит от заданной онтологии и типов входных данных.

Все агенты действуют параллельно, поэтому особенно важно верифицировать некоторые свойства системы, связанные с этим. В частности, интересны свойства корректности определения завершения работы системы агентом-контролером и "работоспособность" системы анализа. Для проверки этих свойств мы использовали инструмент проверки моделей SPIN [9]. SPIN имеет достаточно выразительный входной язык для спецификации нашей модели анализа данных и её свойств и хорошо развитую систему обнаружения и анализа ошибок.

Мультиагентный подход к извлечению информации из разнородных источников данных распространён довольно широко. В частности, он используется в обработке текстов на естественном языке [1, 2, 6, 11] и обработке данных из сети Интернет [3–5]. Агенты обработки текстов и сетевых данных имеют различное поведение. Как правило, при обработке сетевых данных агенты являются высокоуровневыми сущностями, которые скорее управляют потоками данных и используют стандартные алгоритмы извлечения информации. При обработке текстов на естественном языке агенты обычно ассоциируются с традиционными лингвистическими уровнями (морфологическим, синтаксическим, семантическим), либо предназначены распознавать специфические лингвистические феномены, такие как эллипсис, анафора, parataxis, омонимия. Такие агенты могут ускорить процесс анализа данных за счет параллельной работы, но для улучшения качества анализа они не используют онтологические данные содержательно.

Мультиагентный низкоуровневый анализ данных, в котором агенты не обрабатывают входные данные традиционными способами, а непосредственно представля-

ют информационные единицы, является новым методом в области обработки данных. Насколько нам известно, подобный подход представлен только в статье [10]. О верификации подобных систем анализа данных нам также ничего не известно.

Оставшаяся часть статьи организована следующим образом. В Разделе 2 описаны агенты нашей системы и протоколы их действий. Раздел 3 посвящен методу представления данной мультиагентной модели и её свойств в системе SPIN. В заключении 4 обсуждаются направления будущих исследований.

2. Агенты и протоколы

Кратко опишем наш мультиагентный подход к пополнению онтологий на основе семантического анализа данных. Пусть имеются: онтология предметной области, правила её пополнения, семантическая и синтаксическая модель подъязыка предметной области и данные, из которых извлекается информация для пополнения заданной онтологии.

Пусть *онтология предметной области* — это набор $O = \langle C_O, R_O, T_O, A_O \rangle$, где

• $C_O = \cup C_i$ — конечное непустое множество классов, описывающих понятия предметной области;

• $R_O = \cup R_i$ — конечное множество бинарных отношений на классах и $F_R : C_O \times C_O \to 2^{R_O}$ — функция имен отношений между классами;

• $T_O = \cup T_i$ — множество типов данных, где $\{v_1, \ldots, v_i\}$ – область допустимых значений типа T_i;

• $A_O = \cup a_i$ — конечное множество атрибутов, $AK \subseteq A_O$ — подмножество ключевых атрибутов для уникальной идентификации экземпляров понятий и отношений, и $F_A : C_O \cup R_O \to 2^{A_O \times T_O}$ — функция, определяющая имена и типы атрибутов для классов C_O и отношений R_O.

Информационный контент онтологии O — это пара $IC_O = \langle I_O, RI_O \rangle$, где

• $I_O = \cup I_i$ — конечное множество экземпляров классов онтологии O, где I_i класса $C_i \in C_O$ представляется набором атрибутов a_j со значениями v_j: $I_i = \cup_j (a_j, v_j)$, так что $(a_j, v_j) \in F_A(C_i)$;

• $RI_O = \cup RI_i$ — экземпляры отношений онтологии, являющиеся конечными множествами отношений на множестве экземпляров классов I_O, где экземпляр RI_i отношения $R_i \in R_O$ содержит экземпляры $o_1, o_2 \in I_O$ из классов C_1 и C_2 соответственно, а также имеет атрибуты a_j со значениями v_j: $RI_i = ((o_1, o_2), \cup_j (a_j, v_j))$, так что $R_i \in F_R(C_1, C_2)$ и $(a_j, v_j) \in F_A(R_i)$.

Правила для пополнения онтологии распознают во входных данных экземпляры классов или отношений заданной онтологии, означивают их атрибуты и связывают экземпляры классов в отношения. Семантико-синтаксическая модель языка входных данных обычно довольно сложна и её универсальная формализация выходит за рамки данной работы.

Предварительная фаза обработки данных выполняется внешним модулем-анализатором, основанным на словаре предметной области. Этот модуль порождает множества *агентов-экземпляров*, соответствующих понятиям онтологии, и *агентов-отношений*, соответствующих отношениям данной онтологии. *Агенты-правила* реализуют правила обработки входных данных и пополнения онтологий. В соответствии с информацией, полученной от агентов-экземпляров и отношений, они вырабатыва-

ют новые значения элементов экземпляров понятий и отношений и создают новых информационных агентов. Рано или поздно информационные агенты определят все значения своих атрибутов, которые можно извлечь из имеющихся данных, и система остановится. *Агент-контролер* определяет момент остановки системы. Агенты-экземпляры используют накопленные значения каждого из своих атрибутов для разрешения информационных неоднозначностей. Далее следуют формальные определения агентов. Пусть задана онтология $O = \langle C_O, R_O, T_O, A_O \rangle$.

Множество *агентов-экземпляров* IA соответствует экземплярам классов онтологии. Каждый $I \in IA$ является набором $I = (id; Cl_O; Atr; Rul; Rel)$, где

- id — уникальный идентификатор агента;
- $Cl_O \in C_O$ — онтологический класс агента;
- $Atr = \bigcup_{j \in [1..k]}(a_j, V_j, Rul_j)$ — множество атрибутов агента, где для каждого $j \in [1..k]$ (1) a_j — имя атрибута; (2) значения атрибута из V_j принадлежат домену соответствующего типа, так что $(a_j, V_j) \subseteq F_A(Cl_O)$; (3) агентам-правилам из множества Rul_j для получения результата требуется значение атрибута a_j;
- Rul — агенты-правила, требующие данные агента для получения результата;
- Rel — отношения агента; для каждого $(r, ir) \in Rel$: ir — множество идентификаторов экземпляров агента-отношения r, содержащие данного агента.

Множество *агентов-отношений* RlA соответствует отношениям онтологии O. Каждый $Rl \in RlA$ является набором $Rl = (id; Rl_O; IR; Rul)$, где

- id — уникальный идентификатор агента;
- $Rl_O \in R_O$ — онтологическое отношение агента;
- $IR = \bigcup_i((o_1, o_2)_i, Atr_i)$ — множество экземпляров отношения Rl_O, где для каждого i (1) объекты отношения o_1 и o_2 являются идентификаторами агентов-экземпляров классов C_1 и C_2 соответственно, таких что $Rl_O \in F_R(C_1, C_2)$, (2) Atr_i — множество атрибутов отношения и их значений, так что $Atr_i \subseteq F_A(Rl_O)$; агент-отношение определен, если определены его объекты;
- Rul — агенты-правила, требующие данные агента для получения результата.

Множество *агентов-правил* RA соответствует правилам входных данных и пополнения онтологий. Каждый агент $R \in RA$ представляет собой следующий набор $R = (id; Args; make\_res(args), result)$, где

- id — уникальный идентификатор агента;
- $Args = \cup(arg_1(Cl_1), ..., arg_s(Cl_s))$ множество векторов аргументов, где для каждого $i \in [1..s]$: arg_i — значение аргумента, определяемое агентом-экземпляром или отношением онтологического класса $Cl_i \in C_O$; обозначим вектор значений аргументов как $args$, где каждое значение является (1) значением атрибута, снабженного идентификатором соответствующего агента-экземпляра, либо (2) идентификатором агента-экземпляра, либо (3) идентификатором экземпляра агента-отношения;
- $make\_res(args)$ — функция, вычисляющая результат по вектору $args$;
- $result$ результат функции $make\_res(args)$, который может быть пустым, если вектор аргументов не согласован, или содержать (1) значения атрибутов агентов-экземпляров из вектора $args$ и/или (2) объекты и значения атрибутов агентов-отношений из вектора $args$ и/или (3) новых информационных агентов (они отличаются от остальных агентов классами и значениями атрибутов).

Дадим краткий обзор способов взаимодействия информационных агентов и агентов-правил. Мультиагентная система **MDA** для анализа данных содержит мно-

жества информационных агентов, множество агентов-правил и служебного агента контролера. В процессе взаимодействия этих агентов по протоколам, изложенным ниже, осуществляется семантический анализ данных, когда информационные агенты определяют все возможные значения своих атрибутов и объектов, основываясь на входных данных. Все агенты исполняют свои протоколы параллельно. Таким образом агенты действуют до тех пор, пока не окажется, что ни один агент-правило не может выработать результат. Это событие завершения отслеживается агентом-контролером. Мы используем оригинальный AB-алгоритм обнаружения завершения [8], основанный на подсчете активности агентов. Система является динамической, поскольку агенты-правила могут создавать новых информационных агентов.

Агенты связаны двусторонними каналами. Агент-контролер связан со всеми остальными агентами, агенты-экземпляры связаны со своими агентами-отношениями из множества Rel, и все информационные агенты связаны со всеми агентами-правилами, которые используют полученную от них информацию и/или вырабатывают для них новые значения атрибутов/объектов. Мы считаем, что сообщения передаются мгновенно в надежной среде и хранятся в каналах связи до прочтения.

Пусть $IA = \{I_1, ..., I_n, ...\}$ будет множеством агентов-экземпляров с протоколом действий `Ii` для всех $I_i \in IA$, $RlA = \{Rl_1, ..., Rl_m, ...\}$ — множеством агентов-отношений с протоколом действий `Rlj` для всех $Rl_j \in RlA$, и $RA = \{R_1, ..., R_s\}$ — множеством агентов-правил с протоколом действий `Rk` для всех $R_k \in RA$. Пусть `C` — протокол действий агента-контролера C. Тогда алгоритм мультиагентного анализа данных MDA может быть представлен на псевдокоде следующим образом:

`MDA::`

```
parallel {I1} ...{In} ...{Rl1} ...{Rlm} ...{R1} ...{Rs} {C}
```

Здесь оператор `parallel` означает, что все процессы в фигурных скобках исполняются параллельно. Приведём краткие описания протоколов.

Пусть далее C — агент-контролер, R и R_{ij} — агенты-правила, I — агент-экземпляр, Rl — агент-отношение, A — информационный агент, `mess` — сообщение (специализированное для каждого вида агентов), $Input$ — множество входящих сообщений. Для простоты мы предполагаем, что агенты-правила вырабатывают ровно одно значение атрибута для каждого агента-экземпляра и/или ровно один экземпляр для каждого агента-отношения. Протоколы легко можно обобщить на случай выработки множественного результата.

Неформальное описание протокола агента-экземпляра. На первой стадии своей деятельности агент рассылает уже означенные данные всем агентам-правилам, заинтересованным в этих данных. Обработка входных данных агентом-экземпляром заключается в изменении своих атрибутов и отношений, рассылке новых значений агентам-правилам, заинтересованным в этих данных. Агент-контролер информируется о каждом изменении активности. Агент-экземпляр заканчивает работу, если получает сообщение СТОП от агента-контролера.

Protocol of instance agents.

I::

```
1.  send |Rul| + 1 to C;
2.  forall R ∈ Rul send id to R;
3.  forall a_i ∈ Atr
4.      if a_i ≠ ∅ then { send |Rul_i| to C;
```

```
5.        forall R_ij ∈ Rul_i send a_i to R_ij;}
6.  send −1 to C;
7.  while (true){
8.   if Input ≠ ∅ then {
9.      mess = get_head(Input);
10.     if mess.name = C then break;
11.     if mess.name ∈ Rel then upd_Rel(mess.name, mess.id);
12.     if mess.id = i then {
13.        upd(a_i, mess.value);
14.        send |Rul_i| to C;
15.        forall R_ij ∈ Rul_i send a_i to R_ij; }
16.     send −1 to C; } }
```

Неформальное описание протокола агента-отношения. На первой стадии своей деятельности агент рассылает уже означенные данные всем агентам-правилам и агентам-экземплярам, заинтересованным в этих данных. Обработка входных данных агентом-отношением заключается в изменении своих экземпляров, рассылке идентификаторов новых экземпляров агентам-экземплярам и агентам-правилам заинтересованным в этих данных. Агент-контролер информируется о каждом изменении активности. Агент-отношение заканчивает работу, если получает сообщение СТОП от агента-контролера.

Protocol of relation agents.

```
Rl::
1.  send 1 to C;
2.  forall ir_i ∈ IR
3.    if evaluated(ir_i) then {
4.       send |Rul| + 2 to C;
5.       send (Rl, ir_i) to (o_1)_i and (o_2)_i;
6.       forall R ∈ Rul send (Rl, ir_i) to R;}
7.  send −1 to C;
8.  while (true){
9.    if Input ≠ ∅ then {
10.      mess = get_head(Input);
11.      if mess.name = C then break;
12.      upd_Rel(mess.id, mess.value);
13.      i = mess.id
14.      if evaluated(ir_i) then {
15.         send |Rul| + 2 to C;
16.         send (Rl, ir_i) to (o_1)_i and (o_2)_i;
17.         forall R ∈ Rul send (Rl, i) to R;}
18.         send −1 to C; }}
```

Неформальное описание протокола агента-правила. У него есть два параллельных подпроцесса: обработка данных от информационных агентов (`ProcInput`) и получение исходящего результата (`ProcResult`). Обработка входных данных включает (1) формирование вектора аргументов, (2) отправку вектора аргументов либо сообщения об остановке процессу `ProcResult`. Получение исходящего результата включает (1) проверку согласованности аргументов и вектора аргументов, (2) выработку

результата, который может содержать новые значения атрибутов для информационных агентов и/или новых информационных агентов, и (3) определение информационных агентов-получателей. Новые информационные агенты немедленно начинают работать с данными, определенными агентом-правилом при их рождении. Агент-контролер информируется о каждом изменении активности. Агент-правило заканчивает работу, если получает сообщение СТОП от агента-контролера.

Protocol of rule agents.

```
R ::
  SendList: set of Instance Agents = ∅;
1.  parallel
2.   { ProcInput_R; ProcResult_R; }
ProcInput_R ::
  args: set of vectors of Argument;
1.   while (true) {
2.      if Input ≠ ∅ then {
3.        mess = get_head(Input);
4.        if mess.name=C then {
5.           send 'stop' to ProcResult_R; break; }
6.        if mess.name=A then {
7.           args = make_arg(mess.value, A);
8.           if (args ≠ ∅) send ( args ) to ProcResult_R;
9.           send |args| − 1 to C; }}}
ProcResult_R ::
  arg: vector of Argument∪{'stop'};
1.   while (true) {
2.      if Input ≠ ∅ then {
3.        arg = get_head(Input);
4.        if arg = 'stop' then break;
5.        (result, SendList) = make_res(arg);
6.        if result ≠ ∅ then {
7.            start_new_information_agents;
8.            send |SendList| to C;
9.            forall A ∈ SendList send result(A) to A;}
10.       send −1 to C; }}
```

Основная работа агента-контролера заключается в последовательном подсчете активности других агентов. Если все агенты, кроме него самого, неактивны, он посылает всем сообщение СТОП.

Protocol of agent-controller C.

```
C ::
  Act: integer; Input: set of integer;
1.   Act = 0;
2.   while( Input = ∅ ) { }
3.   while(true){
4.    if( Input ≠ ∅ ) then Act = Act + get_mess(Input);
5.    if( Input = ∅ and Act = 0 ) then break; }
6.   send STOP to all;
```

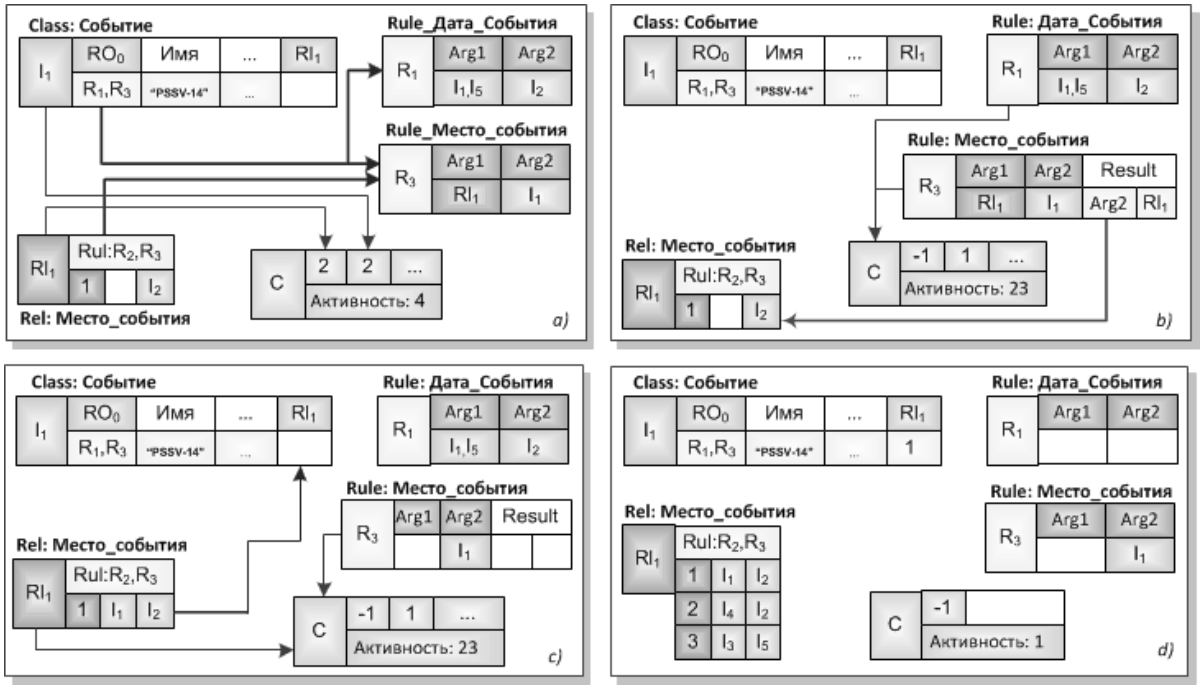

Рис. 1. Иллюстративный пример работы мультиагентного алгоритма анализа текста

Пример на Рис. 1 демонстрирует основные операции, производимые агентами для извлечения информации о месте проведения события, например, из текста информационного письма конференции PSSV. Выделены четыре возможных состояния системы, возникающих последовательно, в которых информационные агенты *События* I_1 (*конференция PSSV-14*) и *Места_событий* Rl_1 отправляют свои данные агенту-правилу R_3, отвечающему за проверку совместимости данных с учетом их выражения в тексте *a)*, и, в случае успеха, получают новые данные *b)*, согласованные друг с другом *c)*. На каждой стадии агент-контроллер C получает информацию об активности агентов и отслеживает момент завершения работы системы *d)*.

Следующее утверждение является прямым следствием утверждения 1 из [7]:

Утверждение 1. *Мультиагентная система* **MDA** *останавливается и агент-контролер корректно определяет момент остановки.*

Утверждение доказано в [7]. Доказательство первой части основано на конечности входных данных и естественном предположении, что правила пополнения онтологии и обработки данных не могут бесконечно порождать новую информацию.

3. Мультиагентный анализ данных в системе SPIN

В дополнение к утверждению 1 имеет смысл формально верифицировать протоколы нашей системы, поскольку параллельное взаимодействие агентов является сложным процессом. Важным свойством данной мультиагентной системы является корректность действий агента-контролера, т.е. то, что этот агент корректно определяет момент остановки системы, когда все агенты находятся в режиме ожидания сообщений, не делая ничего другого. Кроме того, существенным является свойство

работоспособности системы (operability): в будущем наступит момент, когда хотя бы один информационный агент получит и присвоит значение хотя бы одному своему атрибуту. Свойства полноты и точности обработки информации также важны, но их практически невозможно проверить методами формальной верификации.

Для формальной проверки модели нашей мультиагентной системы анализа данных мы использовали известную систему проверки моделей SPIN [9]. Система проверки моделей NuSMV оказалась не подходящей для верификации нашей модели, поскольку входной язык NuSMV не позволяет эффективно работать с массивами, в результате чего описание мультиагентной системы анализа данных оказывается очень сложным. Для верификации с помощью SPIN необходимо, чтобы модель системы была записана на входном языке Promela, а свойства модели могли быть выражены в логике линейного времени LTL.

SPIN допускает только конечные входные данные. Это является существенным основанием для следующего упрощения исходной модели анализа данных: (1) в качестве входных данных мы рассматриваем конечные множества натуральных чисел из ограниченного подмножества натуральных чисел; (2) значения атрибутов классов и экземпляров отношений онтологии, т.е. области допустимых значений, являются множествами натуральных чисел; (3) таким образом, результатом действий агентов-правил являются наборы натуральных чисел в качестве объектов экземпляров агентов-отношений, значений атрибутов для информационных агентов, а также значений элементов новых информационных агентов. Следовательно, в этой упрощенной модели не играет роли, какие именно данные обрабатываются и производятся агентами. Мы заинтересованы в верификации остановки, работоспособности и правильности определения завершения. Основанием для данного упрощения служит первая часть утверждения 1 об остановке мультиагентной системы.

Для спецификации модели в языке Promela определим процессы *InstAgent*, *RelAgent*, *RulAgent* и *Controller*, соответствующие агентам нашей модели. Агенты являются экземплярами процессов соответствующего типа. SPIN сопоставляет каждому процессу уникальный идентификационный номер *_pid*. Далее, мы опишем некоторые характеристики этих процессов.

(1) Определение процессов. Определение процессов основано на формальных описаниях агентов из Раздела 2 и содержат структуры языка Promela, включающие поля, имеющие тип массивов целых чисел. Например, следующая спецификация является частью определения агента-экземпляра:

```
proctype ins_agent(){
    byte id;
    d_step{
        INS_AGENT[INS_AGENT_COUNT] = _pid;
        INS_AGENT_COUNT = INS_AGENT_COUNT + 1;
        id = INS_AGENT_COUNT; }          // unique agent identifier
    int Class;                          // class of the agent
    int RuleOut[MAX_RULE_OUT];          // rules Rul
    Attribute attrs[MAX_ATTR];          // attributes of the agent
    Relation Relations[NUM_INS];        // relations of the agent
    MessagetoRule toRule; ...
```

(2) Типы коммуникационных сообщений. Типы сообщений различны для

различных типов процессов и также реализованы как структуры языка Promela с полями, содержащими целые числа и их массивы. Ниже представлены типы сообщений для агента-экземпляра и агента-правила.

```
typedef MessagetoIns{
 int name;          // name of the sender
 int id;            // name of the relation instance (if any)
 int vals_id;       // name of the attribute (if any)
 int vals_value; }   // value of the attribute (if any)
typedef MessagetoRule{
 mtype type;        // { Agent, Attribute, Relation }
 int name;          // name of the sender
 int val; }           // attribute value or relation instance name
```

(3) Инициализация агентов. Мы создаем набор информационных агентов и назначаем начальные значения их атрибутов и объектов (в случае отношений), что имитирует работу внешнего модуля предварительного анализа данных. Также при инициализации устанавливаются номера агентов-правил, которым необходимы данные этих информационных агентов. Для агентов-правил мы должны задать онтологические классы аргументов и функции получения результата. Предложенная реализация инициализации зависит от порядкового номера каждого агента. Это число определяет класс агента, его исходящие правила Rul и Rul_i для каждого атрибута a_i (см. определение информационных агентов), и атрибуты, значения которых уже определены. Далее следует часть спецификации агента-экземпляра.

```
 Active = 0;                                        // agent activity
 Class = id;
 for (i : 0 .. id-1 ) { RuleOut[i] = i+1; }     // rules Rul
 for (i : 0 .. 2*id-1) {
    attrs[i].RuleOut[0] = i/2+1;                // rules of attributes
    if
        :: (i%2 == 0) ->
            attrs[i].values[0] = i/2+1;         // values of attributes
            attrs[i].values_count = 1;
        :: else -> skip;
    fi; }
 for (i : 0 .. id-1 ) { Relations[i].name = i+1; }...  // relations
```

(4) Действия агентов. Действия основаны на вышеприведенных протоколах и включают в себя пересылку сообщений и обновление данных. Агенты-правила также создают свои векторы аргументов и вычисляют значения атрибутов для информационных агентов, что моделирует правила пополнения онтологий и обработки данных. Действия информационных агентов и агента-контролера транслируются в язык Promela из протоколов предыдущего раздела почти напрямую. Функция $make\_arg$ агента-правила определяет местоположение входных данных, полученных от информационных агентов, в векторе атрибутов в соответствии с определением данного агента-правила. Эта функция формирует очередные данные (множество векторов аргументов) для последующей обработки функцией $make\_res$, которая

имитирует получение результата по входным данным. В нашем случае эта имитация зависит от значений входящих аргументов и $\_pid$ процесса-правила. Эти параметры используются для того, чтобы определить (1) согласованность вектора аргументов; (2) количество и номера агентов-экземпляров, их атрибуты, которые нужно обновить и значения этих атрибутов; (3) количество и номера агентов-отношений, их экземпляры, которые должны быть обновлены или добавлены, изменяемые элементы этих экземпляров и их новые значения. Все имитационные функции очень просты, поскольку предназначены симулировать реальные функции, имеющие линейную временную сложность. Однако заметим, что функция $make\_arg$ имеет экспоненциальную временную сложность. Представленная ниже часть спецификации агента-правила вычисляет новое значение атрибута агента-экземпляра.

```
proctype rule_agent(){                  // start consuming subprocess
    byte id;                            // unique agent identifier
    ... ...
    run rule_agent_out(_pid, id);       // start producing subprocess
 ... ...
   for(i: id .. 2*id-1){
    mti.vals_id = id+1;                          // updated attribute
    mti.vals_value = argV[i].val;                // new attribute value
    toController ! ( 1 );                        // info for controller
    toInsAgent[ argV[i].name ] ! ( mti );        // send the new value
   } ... ...
```

Сформулируем свойства модели, которые мы хотим верифицировать. Пусть у каждого агента $A \in IA \cup RlA \cup RA$ (не контролера) есть специальный булевский статус активности $A.active$, и его значение истинно, когда агент делает что-нибудь полезное (посылает или обрабатывает сообщения), и ложно, когда агент просто ожидает входящие сообщения. Тогда свойство корректности работы контролера можно выразить в логике линейного времени LTL следующим образом:

$$G(Act = 0 \rightarrow \bigwedge_{A \in IA \cup RlA \cup RA} A.active = false).$$

Свойство работоспособности выражается так:

$$F(\bigvee_{A \in IA \cup RlA} A.was\_upd = true),$$

где $A.was\_upd$ — булевская переменная, отражающая тот факт, что агент A обновил свой атрибут, т.е. изначально значение этой переменной равно $false$, и после первого обновления атрибута оно становится равным $true$.

4. Заключение

В данной статье мы предложили подход к верификации мультиагентного алгоритма анализа данных для пополнения онтологии. Средством верификации служит

инструмент проверки моделей SPIN, а свойства системы выражены с помощью формул логики линейного времени LTL. В режиме симуляции модель системы проверена для 150 основных агентов и агента-контролера (138130 шагов). Однако в режиме верификации проверка свойств корректности оказалась возможной только для 18 основных агентов. Оба свойства корректности выполняются в нашей модели. Для верификации потребовалось 25 минут работы компьютера с процессором класса Intel Celeron(R) с тактовой частотой 2.6 ГГц и около 1 ГБайта оперативной памяти.

На данном этапе наших исследований мы не рассматриваем конкуренцию и кооперацию информационных агентов для разрешения неоднозначностей. В ближайшем будущем мы планируем расширить возможности агентов данными типами взаимодействий, разработать алгоритмы для разрешения неоднозначностей и верифицировать такие их свойства, как завершение, корректность взаимодействий и т.п.

Список литературы

1. Aref M.M. *A Multi-Agent System for Natural Language Understanding* // International Conference on Integration of Knowledge Intensive Multi-Agent Systems. 2003. 36.

2. A.M.B.R. Carvalho, D.S. de Paiva, J.S. Sichman, J.L.T. da Silva, R.S. Wazlawick & V.L.S. de Lima *Multi-Agent Systems for Natural Language Processing* // Multi Agent Systems Models Architecture and Appications: Proceedings of the II Iberoamerican Workshop on D.A.I. and M.A.S / Francisco J. Garijo & Cristian Lemaitre (eds.) Toledo, Spain, October 1–2 1998. P. 61–69.

3. Banares-Alcantara R., Jimenez R., Aldea L. *Multi-agent systems for ontology-based information retrieval* // European Symposium on Computer-Aided Chemical Engineering-15 (ESCAPE-15). Barcelona, Espana, 2005.

4. Cheng X., Xie Y., Yang T. *Study of Multi-Agent Information Retrieval Model in Semantic Web* // Proc. of the 2008 International Workshop on Education Technology and Training and 2008 International Workshop on Geoscience and Remote Sensing (ETTANDGRS'08). 2008. Vol. 02. P. 636–639.

5. Clark K.L., Lazarou V.S. *A Multi-Agent System for Distributed Information Retrieval on the World Wide Web* // Proc. of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises. 1997. P. 87–93.

6. D. Fum, G. Guida, C. Tasso *A Distributed Multi-Agent Architecture for Natural Language Processing* // Proc. of the 12th conference on Computational linguistics (COLING '88). 1988. Vol. 2. P. 812–814.

7. Garanina N., Sidorova E., Bodin E. *A Multi-agent Approach to Unstructured Data Analysis Based on Domain-specific Onthology* // Proc. of the 22nd International Workshop on Concurrency, Specification and Programming. Warsaw, Poland, Sept. 25–27, 2013. CEUR Workshop Proceedings, Vol. 1032. P. 122–132.

8. Garanina N., Bodin E. *Distributed Termination Detection by Counting Agent* // Proc. of the 23nd International Workshop on Concurrency, Specification and Programming. Chemnitz, Germany, 29 September – 01 Oktober 2014. Humboldt-Universität zu Berlin, 2014. P. 69–79.

9. Holzmann G.J. The Spin Model Checker: Primer and Reference Manual // Addison Wesley Pub, 2003. P. 608.

10. Minakov I., Rzevski G., Skobelev P., Volman S. *Creating Contract Templates for Car Insurance Using Multi-agent Based Text Understanding and Clustering* // Proc. Holonic and Multi-Agent Systems for Manufacturing, Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2007. Regensburg, Germany, September 3–5, 2007. Springer, Lecture Notes in Computer Science, 2007. Vol. 4659. P. 361–370.

11. C.T. dos Santos, P. Quaresma, I. Rodrigues, R. Vieira *A Multi-Agent Approach to Question Answering* // Computational Processing of the Portuguese Language: 7th International Workshop, PROPOR 2006. Itatiaia, Brazil, May 2006 (PROPOR'2006). LNAI 3960, 13–17 de Maio de 2006. Berlin/Heidelberg: Springer Verlag. P. 131–139.

Using SPIN for Verification of Multi-agent Data Analysis

Garanina N.O., Bodin E.V., Sidorova E.A.

A.P. Ershov Institute of Informatics Systems, Russian Academy of Sciences Siberian Branch, Acad. Lavrentjev pr., 6, Novosibirsk, 630090, Russia

Keywords: ontology population, multi-agent system, model checking, SPIN

The paper presents an approach to formal verification of multi-agent data analysis algorithms for ontology population. The system agents correspond to information items of the input data and the rule of ontology population and data processing. They determine values of information objects obtained at the preliminary phase of the analysis. The agents working in parallel check the syntactic and semantic consistency of tuples of information items. Since the agents operate in parallel, it is necessary to verify some important properties of the system related to it, such as the property that the controller agent correctly determines the system termination. In our approach, the model checking tool SPIN is used. The protocols of agents are written in Promela language (the input language of the tool) and the properties of the multi-agent data analysis system are expressed in the liner time logic LTL. We carried out several experiments to check this model in various modes of the tool and various numbers of agents.

Сведения об авторах:
Гаранина Наталья Олеговна,
Институт систем информатики им. А.П. Ершова СО РАН,
старший научный сотрудник,
Бодин Евгений Викторович,
Институт систем информатики им. А.П. Ершова СО РАН,
научный сотрудник,
Сидорова Елена Анатольевна,
Институт систем информатики им. А.П. Ершова СО РАН,
старший научный сотрудник.