Модели и методы представления знаний в CASE-технологии

В.Н. Вагин, Е.Ю. Головина, Ф.Ф. Оськин

Рассмотрен новый подход к созданию репозитория, основанный на концепциях "программной инженерии"и "инженерии знаний". Предложена гибридная модель предметной области CASE-системы. Приведены механизмы обработки знаний в гибридной модели. Разработана система КМ моделирования сложноструктурированной проблемной области.

1 Введение

Широкое использование вычислительной техники в различных сферах деятельности человека привело к потребности создания соответствующего программного обеспечения (ПО). Однако трудоемкость и наукоемкость разработки программ настолько огромны, что ведутся работы по созданию новых технологий автоматизации проектирования программных средств. Это направление получило название CASE-технология (Computer-Aided Software Engineering) [1].

Один из центральных звеньев в CASE-системе это репозиторий (информационная база проекта), являющийся основой интеграции в технологических системах. В репозитории сосредоточена информация о создаваемом ПО на всех стадиях жизненного цикла (ЖЦ) от технического задания до сопровождения [1].

Репозиторий, построенный на основе традиционного подхода, представляет собой хранилище информации, необходимой для разработки ПО в CASE-системе.

В настоящее время возникла необходимость в создании интеллектуального репозитория, который облегчит разработчикам процесс создания в CASE-системе ПО, отвечающего современному уровню.

Создание интеллектуального репозитория имеет ряд преимуществ по сравнению с традиционным подходом к его разработке, как БД и система управления БД (СУБД), среди которых можно выделить следующие:

- возможность представления знаний, которые являются обобщением накопленного опыта о процессе проектирования ПО, и предоставление доступа к этим знаниям при разработке ПО;
- получение новых знаний о разрабатываемом ПО из знаний, представленных в репозитории, которые помогают разработчику в процессе его проектирования;
- возможность контроля непротиворечивости знаний;
- возможность прогнозирования результатов проектных операций;
- возможность сократить объем информации, представленной в репозитории, благодаря ее получению с помощью логического вывода и процедур.

Компонентами интеллектуального репозитория являются база знаний (БЗ), интеллектуальная информационно-поисковая система, подсистема помощи при проектировании ПО, вспомогательные компоненты [8].

Вся информация о разрабатываемом ПО, которая может быть формализована, представляется в модели предметной области CASE-системы. Например, модель предметной области CASE-системы для создания систем поддержки управления состоит из следующих подмоделей: модели проектируемого ПО, модели организации пользователя, модели наблюдаемого (внешнего) объекта и системы наблюдения (контроля).

2 Гибридная модель предметной области CASE-системы

Поскольку в основе БЗ лежит формальная модель предметной области CASE-системы, рассмотрим метод ее построения. В качестве формальной модели предметной области CASE-системы выбираем гибридную модель, объединяющую парадигмы "программная инженерия"и "инжене-

рия знаний": объектно-ориентированный подход [9, 10, 11], процедуры и модель представления знаний предметной области.

В качестве формализма представления знаний предметной области CASE-системы предлагаем использовать логическую модель, основанную на многоуровневой логике (Multi-layer logic или коротко MLL), разработанную Setsuo Ohsuga и Hiroyuki Yamauchi [12, 13], которая является удобным аппаратом для формализации сложноструктурированной информации, выделяемой в процессе проектирования ПО. MLL является интеграцией логического подхода и подхода, основанного на семантической сети, к построению языка представления знаний. Процедуры вызываются в процессе дедуктивного вывода. Они используются, например, для определения экстенсионалов отношений , нахождения значений атрибутов, реализации операций над объектами.

Слэш - некоторый разделитель, который используется в префиксе логической формулы. Так, простой слэш (Qx/X) используется для обозначения, что x является элементом множества X (x∈X), простой "жирный"слэш (Qx/X) обозначает, что x определен на множестве, элементами которого являются множество частей объекта одного типа (X▷x), двойной слэш (Qx// X) обозначает, что x определен на множестве, элементами которого являются части объекта X (X♢x), где Q = {∀,∃} [12]. Отношение ♢ является композицией двух отношений ▷ и ∈, т.е. ♢ = ∈∘▷.

Если объект Y имеет в качестве компонент несколько объектов, то для того, чтобы задать нужную компоненту X объекта Y, необходимо использовать селектор, который представляется предикатом F(X,Y) [12]. Поэтому, чтобы определить свойства x ∈ X, который является частью объекта Y, необходимо написать формулу :

$$(\exists X/\#\mathrm{Y})(\mathrm{Qx/X})[\mathrm{F(X,Y)\&G(x)}],$$

которая может быть преобразована к стандартной форме [12]:

$$(Qx//\#Y)[F(\mathrm{x},Y)\&G(x)],$$

где , Q = {∀,∃}, # - обозначение константы, G(x) - описывает свойства x.

Нами развит синтаксис MLL за счет введения расширения, которое позволяет заменить селектор, используемый для нахождения нужной

компоненты некоторого объекта, на композицию префикса логической формулы.

Пусть объект #Y имеет в качестве компонент несколько объектов, т.е. <#Y>=$\{X_i\}$, i = 1,N. Тогда префикс может содержать запись вида:

(Q(x/X_i)//#Y), (1)

где Q=$\{\forall,\exists\}$, # - обозначение константы. X_i задает сортность x. #Y будем называть $\diamondsuit$- предком для x по иерархии Part of.

Запись (1) содержательно означает, что x определена на объединении частей #Y, которые имеют сортность X_i.

Преимущества такого подхода заключаются в том, что :

- во-первых - не затрачивается время на унификацию предиката - селектора, расширенный нами синтаксис префикса логической формулы позволяет сделать означивание термов по структурам проблемной области и далее осуществить проверку значений переменных на удовлетворение условию, задаваемому матрицей логической формулы;

- во-вторых - не затрачивается память на хранение многочисленных фактов проблемной области для означивания переменных в предикате-селекторе.

Из вышерассмотренного следует, что предложенное расширение синтаксиса MLL повышает эффективность дедуктивного вывода по памяти и быстродействию.

Расширенный синтаксис MLL имеет следующий вид.

Алфавит :

(1). константы: a,b,c,...,X,Y,Z (константные множества),...

(2). переменные: x,y,z,...

(3). функциональные символы: f,g,h,...

(4). предикатные символы: P,Q,R,...

(5). кванторы: $\forall, \exists$

(6). отрицание: $\neg$

(7). логические связки: &, $\vee$,

(8). вспомогательные символы: #,*,/,/,//,,,(,)

Термы.

(1). Любая константа и переменная являются термом.

(2). Если f есть n-местный функциональный символ и $t_1, t_2, ..., t_n$ являются термами, то $f(t_1, t_2, ..., t_n)$ есть терм.

(3). Все термы получаются применением (1) и (2).

Правила образования правильно-построенных формул.

F1. Если P является n-местным предикатным символом и $t_1, t_2, ..., t_n$ есть термы, то $P(t_1, t_2, ..., t_n)$ является ППФ (атомарной формулой).

F2. Если F и G - ППФ, $\neg F, F\&G, F \vee G, F \rightarrow G, F \leftrightarrow G$ являются ППФ.

F3. Если F - ППФ и x - предметная переменная, то

(1). $(\forall x/y)F$ и $(\exists x/y)F$ являются ППФ, где y есть константа или переменная.

(2). $(\forall x/\mathrm{X})\mathrm{F}$ и $(\exists x/\mathrm{X})\mathrm{F}$ являются ППФ , где y есть константа или переменная.

(3). $(\forall x//y)F$ и $(\exists x//y)F$ являются ППФ, где y есть константа или переменная.

(4). $(\forall(x/Z)//y)F$ и $(\exists(x/Z)//y)F$ являются ППФ, где y есть константа или переменная, Z есть константное множество.

F4. Других правил образования ППФ нет.

Приведем пример логического выражения, содержащегося в интенсиональной составляющей БЗ, и на нем покажем преимущества использования аппарата MLL для формализации сложноструктурированной предметной области по сравнению с аппаратом многосортной логики (MSL).

Пример.Программная компонента x, входящая в состав системы поддержки управления #P, обеспечивает посадку самолета y, приписанного к аэропорту #A, если имеется :

- ЭВМ t, на которой функционирует x;
- радио-локационная станция (РЛС) s, соединенная с ЭВМ t;
- поток информации #I1, содержащий поток сообщений , принимаемый РЛС s и содержащий класс сообщений , описывающий самолет y и обрабатываемый x;
- поток информации #I2, содержащий поток сообщений, передающийся РЛС s, в который входит класс сообщений , содержащий

сведения, необходимые для посадки самолета y, вырабатываемый x и принимаемый y.

Запись в MSL :

(#A/Аэропорт)(#P/СПУ)(∃x/программная_подсистема)
(∀y/самолет)(∃s/РЛС)(∃t/ЭВМ)(#I1/поток_информации)
(#I2/поток_информации)(∃t_1/поток_сообщений)
(∃r_1/класс_сообщений)(∃t_2/поток_сообщений)
(∃r_2/класс_сообщений)
Содержит(#P,x) & Содержит(#A,y)&Содержит(#A,s) & Содержит(#A, t) & Функционирует(x,t) & Соединена(s,t) & Содержит(#I1,t_1) & Принимает_РЛС(s,t_1) & Содержит(t_1,r_1) & Описывает(r_1,y) & Обрабатывает(x,r_1) & Содержит(#I2,t_2) & Передает_РЛС(s,t_2) & Содержит(t_2,r_2) & Выраба-тывает(x,r_2) & Принимает_самолет(y, → Обеспечивает_посадку(x,y)

Запись в MLL :

(∃(x/программная_система)//#P)(∀(y/самолет)//#O)
(∃(s/РЛС)//#A)(∃(t/ЭВМ)//#A)
(∃(t_1/поток_сообщений)//#I1)
(∃(r_1/класс_сообщений)//t_1)(∃(t_2/поток_сообщений)//#I2)
(∃(r_2/класс_сообщений)//t_2)
Функционирует(x,t) & Соединена(s,t) & Принимает_РЛС(s,t_1) & Описывает(r_1,y) & Обрабатывает(x,r_1) & Передает_РЛС(s,t_2) & Вырабатывает(x,r_2) & Принимает_самолет(y,r_2) → Обеспечивает_посадку(x,y)

Из вышерассмотренного примера следует, что аппарат MLL увеличивает эффективность дедуктивного вывода по сравнению с MSL за счет сокращения пространства поиска, которое достигается структуризацией проблемной области, и представления структурных отношений в префиксе логической формулы, что влечет удаление большого числа предикатов, которые загромождают описание проблемной области и снижают эффективность дедуктивного вывода.

3 Механизмы обработки знаний в CASE-технологии.

Неотъемлемой частью любой БЗ являются механизмы обработки знаний, присутствие которых является главной отличительной особенностью БЗ

от БД. В настоящее время наиболее широко используемым механизмом обработки знаний является дедуктивный вывод.

Дедуктивный вывод позволяет:

- получать новые знания из знаний, представленных в БЗ;
- осуществить контроль информации, представленной в БЗ, на непротиворечивость;
- "сжать"экстенсиональную составляющую БЗ (за счет получения экстенсионалов атрибутов и отношений при помощи дедуктивного вывода);
- получить ответы на запросы пользователей.

В процессе дедуктивного вывода должна быть предусмотрена возможность вызывать компоненты из процедурной составляющей БЗ для обработки информации, представленной в БЗ.

Рассмотрим дедуктивный вывод в MLL более подробно. В процесс дедуктивного вывода вовлекаются два важных алгоритма алгоритм сколемизации и алгоритм унификации. Эти алгоритмы в MLL являются дальнейшим развитием соответствующих алгоритмов в исчислении предикатов первого порядка.

Исходная формула для алгоритма сколемизации должна быть представлена в пренексной нормальной форме.

3.1 Алгоритм Сколемизации

Рассмотрим особенности алгоритма сколемизации в MLL. Алгоритм сколемизации основывается на следующих положениях.

А. Сколемовская функция включает в качестве аргументов все переменные, которые связаны квантором $\forall$ и которые расположены слева от квантора $\exists$, за исключением переменной, которая является $\diamondsuit$ - предком переменной, связанной квантором $\exists$.

Б. $\diamondsuit$ - предок и домен должны быть включены в сколемовскую функцию и в матрицу для процедуры унификации. Они сохраняются в форме (x/X)//Y.

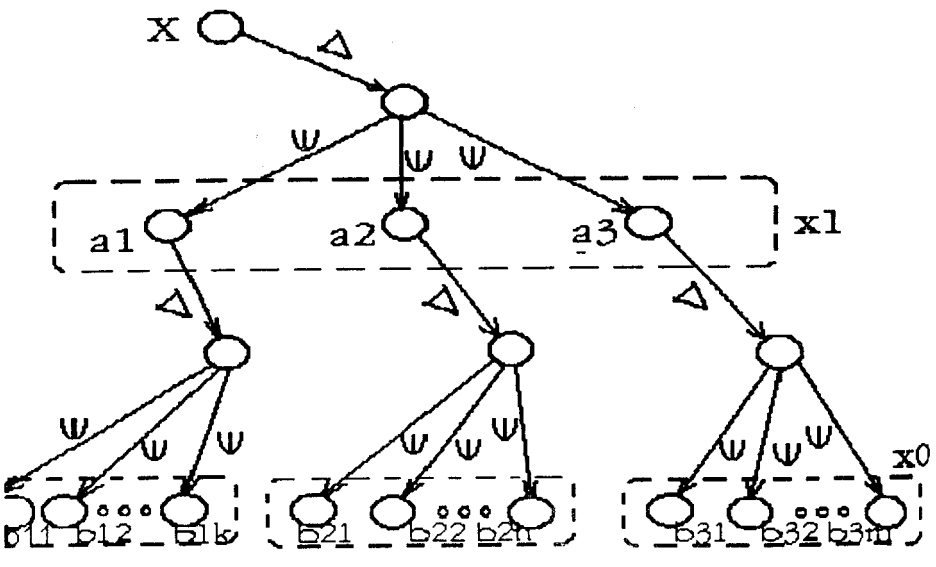
Объясним положение (А) на примере.

Поскольку положение (А) не зависит от числа компонент объектов, рассмотрим случай, когда объекты состоят из одной компоненты. При этом области определения термов однозначно определяются их $\diamondsuit$ - предками по иерархической структуре. Иерархическая структура представлена на рис. 1.

Рассмотрим формулу : $(\forall x_1//\mathrm{X})$ ($\exists x_0//\mathrm{x1})$ $\mathrm{P}(x_1,x_0)$ (2) Формула (2) читается следующим образом : Для любого объекта x_1, который является частью объекта X, (на рис.1. это a_1, a_2, a_3), существует объект x_0, который является частью объекта x_1, (на рис.1. это b_{11} или b_{12} или b_{1k}, если x_1 есть a_1, и b_{21} или b_{22} или b_{2n}, если x_1 есть a_2, и b_{31} или b_{32} или b_{3m}, если x_1 есть a_3), такой что $\mathrm{P}(\mathrm{x}_1,\mathrm{x}_0)$.

В классическом алгоритме сколемизации x_0 заменяется на функцию $f_c(x_1)$, т.е. получим формулу с учетом положения (Б) :

$P(x_1//X,f_{\mathrm{c}}(x_1//X)//(x_1//\mathrm{X}))$.



Поскольку иерархические структуры являются фиксированными, то фиксированными являются и все структурные отношения в ней, и, следовательно, области определения термов.

Поскольку функция $f_c(x_1//\mathrm{X})//(x_1//\mathrm{X})$ определена на множестве частей объекта, заданного переменной x_1 , то она не зависит непосредственно от переменной $\mathrm{x}_1//\mathrm{X}$. Следовательно, функция f_c заменяется на "обобщенную"константу B_c, которой может быть любая константа b_{11} или b_{12} или b_{1k}, если x_1 есть a_1, и b_{21} или b_{22} или b_{2n}, если x_1 есть a_2, и b_{31} или b_{32} или b_{3m}, если x_1 есть a_3, и на которую заменяется переменная x_0 в классическом алгоритме сколемизации, если она связана квантором $\exists$ и стоит на первом месте в префиксе. В результате получим формулу:

$\mathrm{P}(\ x_1//\mathrm{X},\ B_c//(x_1//\mathrm{X})\ )$.

Отсюда следует, что при построении сколемовской функции для переменной x_0 переменная x_1 "мысленно"вычеркивается из префикса.

Это положение может быть распространено на произвольную формулу MLL. Пусть задана формула :

$(\forall x_1//X_1\ )...(\forall x_{n-1}\ //X_{n-1}\ )\ (\exists x_n\ //x_1\ )\ \mathrm{P}(x_1\ ,..,x_n\ )$ (3)

Применим классический алгоритм сколемизации к формуле (3). Получим формулу :

$\mathrm{P}(x_1//X_1,...,x_{n-1}//X_{n-1},\mathrm{f}(x_1//X_1,...,x_{n-1}//X_{n-1})//(x_1//X_1))$

Поскольку f определена на множестве частей объекта, заданного переменной $\mathrm{x}_1//\mathrm{X}_1$, то значение функции f не зависит непосредственно от переменной $\mathrm{x}_1//\mathrm{X}_1$, и следовательно, переменная x_n в алгоритме сколемизации заменяется на функцию $\mathrm{f}(x_2//X_2,...,x_{n-1}\ //\ X_{n-1}\ )//\ (x_1//X_1\ )$.

Рассмотрим алгоритм сколемизации в MLL с учетом введенного расширения синтаксиса.

Алгоритм сколемизации в MLL.

Пусть формула представлена в виде: $(Q_1\ (x_1/X_1)//Y_1)(Q_2\ (x_2/X_2)//Y_2)$ $...(Q_r\ (x_r/X_r)//Y_r)\ \mathrm{P}(x_1,x_2,..,x_r)$, где $\mathrm{Q}=\{\forall,\exists\}$, X_i - сортность переменной x_i, Y_i - $\diamondsuit$ - предок переменной x_i, $\mathrm{i}=1\div\mathrm{r}$, P - бесквантерная формула (или матрица). (В качестве Y_i, $\mathrm{i}=1\div\mathrm{r}$ может быть константа или переменная).

Шаг1.Найти наименьший индекс i такой, что Q_1, Q_2,...,Q_{i-1} все равны $\forall$, а $Q_i = \exists$.

Если такого i нет, то перейти к шагу 4. Если i=1, то x_i заменяется на константу $(@\mathrm{a}/\mathrm{X}_i)//Y_i$ во всех вхождениях в матрицу и на @a во всех вхождениях x_i в префикс в виде $\diamondsuit$ - предка другой переменной.

Если $\mathrm{i}\neq1$, то перейти к шагу 3.

Шаг 2.Квантор $Q_i = \exists$ вычеркивается. Перейти к шагу 1.

Шаг 3.Все переменные с индексом до i сравнить с $\diamondsuit$ - предком i-ой переменной.

Если совпадения нет, то взять новый (i-1)-местный функциональный символ f_i. Переменная x_i заменяется на сколемовскую функцию $(f_i(x_1,x_2, ...,x_{i-1})/X_i)//Y_i$ во всех вхождениях в матрицу и на функцию f_i $(x_1 ,x_2 ,...,x_{i-1})$ во всех вхождениях x_i в префикс, и перейти к шагу 2.

Иначе (т.е. имеется переменная x_j (j < i) под квантором $\forall$, которая является и $\diamondsuit$ - предком заменяемой переменной).

Если i-2 равняется нулю, то x_i заменяется на $(@a/X_i)//x_j$ во всех вхождениях в матрицу и на @a во всех вхождениях в префикс как $\diamondsuit$ - предок другой переменной, и перейти к шагу 2.

Взять (i-2)-местный функциональный символ, не встречавшийся в формуле. Переменная x_i заменяется на сколемовскую функцию $(f_i(x_1,x_2,..., x_{i-2})/X_i)//x_j$ во всех вхождениях в матрицу и на функцию f_i $(x_1 ,x_2 ,...,x_{i-2})$, где $x_k \neq x_j$, k=1÷i-2 во всех вхождениях в префикс как предок другой переменной, и перейти к шагу 2.

Шаг 4.Если кванторов $\forall$ нет, то stop. Если в префиксе формулы в качестве $\diamondsuit$ - предков выступают сколемовские функции, то аргументам функции приписываются их домены и $\diamondsuit$ - предки.

Всем переменным, стоящим под квантором $\forall$, в матрице формулы приписываются их домены и $\diamondsuit$ - предки, за исключениям случая, когда переменная является $\diamondsuit$ - предком другой переменной.

Кванторы $\forall$ опускаются, предполагая, что они существуют стоп. Рассмотрим пример использования алгоритма. Предположим, что объекты X,Y,Z имеют в качестве компонент несколько объектов.

$(\forall(x_1/D_1)//X)(\exists(y_1/D_2)//Y)(\forall(y_0/D_3)//y_1)(\exists(x_0/D_4)//x_1)(\forall(z_1/D_5)//Z)$
$P(x_1,x_0,y_1,y_0,z_1)$

*шаг 1.*i=2

шаг 3.$(\forall(x_1/D_1)//X)(\exists(y_1/D_2)//Y)(\forall(y_0/D_3)//f_1(x_1))$
$(\exists(x_0/D_4)//x_1)(\forall(z_1/D_5)//Z)$
$P(x_1,x_0,(f_1(x_1)/D_2)//Y,y_0,z_1)$

шаг 2.$(\forall(x_1/D_1)//X)(\forall(y_0/D_3)//f_1(x_1))(\exists(x_0/D_4)//x_1)(\forall(z_1/D_5)//Z)$
$P(x_1,x_0,(f_1(x_1)/D_2)//Y,y_0,z_1)$

*шаг 1.*i=3

шаг 3.$(\forall(x_1/D_1)//X)(\forall(y_0/D_3)//f_1(x_1))(\exists(x_0/D_4)//x_1)(\forall(z_1/D_5)//Z)$
$P(x_1,(f_2(y_0)/D_4)//x_1,(f_1(x_1)/D_2)//Y,y_0,z_1)$

шаг 2.$(\forall(x_1/D_1)//X)(\forall(y_0/D_3)//f_1(x_1))(\forall(z_1/D_5)//Z)$

P(x_1,($f_2(y_0)/D_4$)//x_1,($f_1(x_1)/D_2$)//Y,y_0,z_1)
*шаг 1.*i нет
*шаг 4.*P((x_1/D_1)//X, (f_2((y_0/D_3)//f_1((x_1/D_1)//X))/D_4)//x_1,
(f_1((x_1/D_1)//X)/D_2)//Y, (y_0/D_3)//f_1((x_1/D_1)//X), (z1/D5)//Z)
стоп.

3.2 Алгоритм унификации

Рассмотрим особенности алгоритма семантической унификации в MLL :

1. Термы в MLL унифицируются при выполнении условий унификации, которые являются одинаковыми для $\diamondsuit$ - предков и доменов: X$\subseteq$Y и X$\bigcap$Y $\neq 0$.

2. Пусть x является термом уровня i, и в тоже время, он является $\diamondsuit$ - предком терма следующего нижележащего уровня. Терм x унифицируется и автоматически унифицируются $\diamondsuit$ - предок терма следующего нижележащего уровня.

Алгоритм унификации для MLL.

1. Установить k=0, W_k=W, $S_k = \oslash$, где W - унифицируемые выражения и S_k - наибольший общий унификатор.

2. Если W_k является одноэлементным множеством, то S=S_k и stop. В противном случае переход к п.3.

3. Каждая из литер в W_k рассматривается, как цепочка символов и выделяются первые подвыражения литер, не являющихся одинаковыми у всех элементов W_k , т.е. образуется так называемое множество рассогласований типа { (v_k/V_k)//V, (t_k/T_k)//T }.

Если v_k - переменная, а t_k - терм или v_k - функция, а t_k - переменная или функция, то перейти к п.4.

В противном случае стоп: W не унифицируемо.

4. Проверка условий унификации $\diamondsuit$ - предков :

Если V $\subseteq$ T или V $\bigcap$ T = Z ($\neq\oslash$) (условия унификации выполняются), то перейти к п.5.

В противном случае стоп : W не унифицируемо.

Если $\diamondsuit$ - предками являются термы и они совпадают, то перейти к п.5.

В противном случае стоп : W не унифицируемо.

5. Проверка условий унификации доменов :

а) если v_k - переменная, определенная на домене V, а t_k константа, определенная на домене T, и T $\subseteq$ V, то перейти к п.6. В противном случае стоп : W не унифицируемо.

б) если v_k - переменная, определенная на домене V_k, а t_k - переменная, определенная на домене T_k, и $T_k \bigcap V_k = Z_k$ ($\neq\oslash$), то перейти к п.6. В противном случае стоп : W не унифицируемо.

в) если v_k - переменная, определенная на домене V_k, а t_k - функция, определенная на домене T_k, и $T_k \subseteq V_k = Z_k$ ($\neq\oslash$), то перейти к п.6. В противном случае стоп : W не унифицируемо.

г) если v_k - функция, определенная на домене V_k, а t_k функция, определенная на домене T_k, и $T_k \subseteq V_k$ или $V_k \subseteq T_k$,то перейти к п.6. В противном случае stop : W неунифицируемо.

6. Если v_k - переменная, а t_k - константа или функция, то $S_{k+1} = S_k \circ \{t_k \ / \ v_k \ \}$ и $W_{k+1} = W_k \ \{ \ t_k \ / \ v_k \ \}$, где $\circ$ - обозначает добавление $\{t_k \ / \ v_k \ \}$ в S_k; запись $W_k\{ \ t_k \ / \ v_k \ \}$ обозначает замену v_k на t_k в W_k (см. примечание).

Если v_k - переменная, а t_k - переменная, то $S_{k+1} = S_k \circ \{z_k/t_k, \ z_k/v_k \ \}$ и $W_{k+1} = W_k\{z_k \ / \ t_k, \ z_k \ / \ v_k \ \}$, где z_k - переменная, отличная от v_k и t_k.

Если v_k и t_k - функции, то в унифицируемое выражение подставляется функция, область значений которой является подмножеством области значений другой функции.

Примечание. Запись $\{t_k \ /v_k \ \}$ означает замену $(v_k/V_k)//\mathrm{V}$ на $(t_k/T_k)//\mathrm{T}$ во всех вхождениях v_k как терм и замену v_k на t_k во всех вхождениях v_k как $\diamondsuit$ - предка. Аналогично, когда v_k и t_k определены на термах.

7. k=k+1, перейти к п.2.

В случае, когда объекты состоят из одной компоненты, сортность термов не указывается в префиксе и шаг проверки унификации доменов термов в формулах пропускается, поскольку области определения термов однозначно задаются их $\diamondsuit$ - предками.

Рассмотрим пример использования алгоритма унификации.

Пример приведен для случая, когда объекты в качестве компонент имеют несколько объектов. Иерархические структуры проблемной области приведены на рис. 2.

W=$\{$ P($(a_1/A_1)//\mathrm{X}, (x_0/X_0)//a_1, (y_1/Y_1)//\mathrm{Y}$, (f$((x_0/X_0)//a_1)/W_0)//y_1)$, P$((v_1/V_1)//\mathrm{V}$,(g$((w_1/W_1)//\mathrm{W})/X_0)//v_1$,$(w_1/W_1)//\mathrm{W}$, $(w_0/W_0)//w_1)$ $\}$

1. W_0=W; $S_0=\oslash$

2. $\{$ $(a_1/A_1)//\mathrm{X}$, $(v_1/V_1)//\mathrm{V}$ $\}$ - множество рассогласований

3. Полагаем, что X $\subseteq$ V, Полагаем, что $A_1 \subseteq V_1$,

4. S_1=$S_0 \circ \{a_1/v_1\}$ W_1 = { P((a_1/A_1)//X, (x_0/X_0)//a_1, (y_1/Y_1)//Y, (f(((x_0/X_0)//a_1)/W_0)//y_1), P((a_1/A_1)//X, (g(((w_1/W_1)//W)/X_0)//a_1,(w_1/W_1) //W, (w_0/W_0)//w_1) }

5. { (x_0/X_0)//a_1, (g(((w_1/W_1)//W)/X_0)//a_1 } - множество рассогласований.

6. S_2=S_1 ({ g(((w_1/W_1)//W)/x_0 } W_2 = { P((a_1/A_1)//X, (g(((w_1/W_1)//W) /X_0)//a_1, (y_1/Y_1)//Y, (f((g(((w_1/W_1)//W)/X_0)//a_1)/W_0)//y_1), P((a_1/A_1)//X, (g(($(w_1/$ $W_1)$//W)/X_0)//a_1, (w_1/W_1)//W, (w_0/W_0)//w_1) }

7. { (y_1/Y_1)//Y, (w_1/W_1)//W } - множество рассогласований

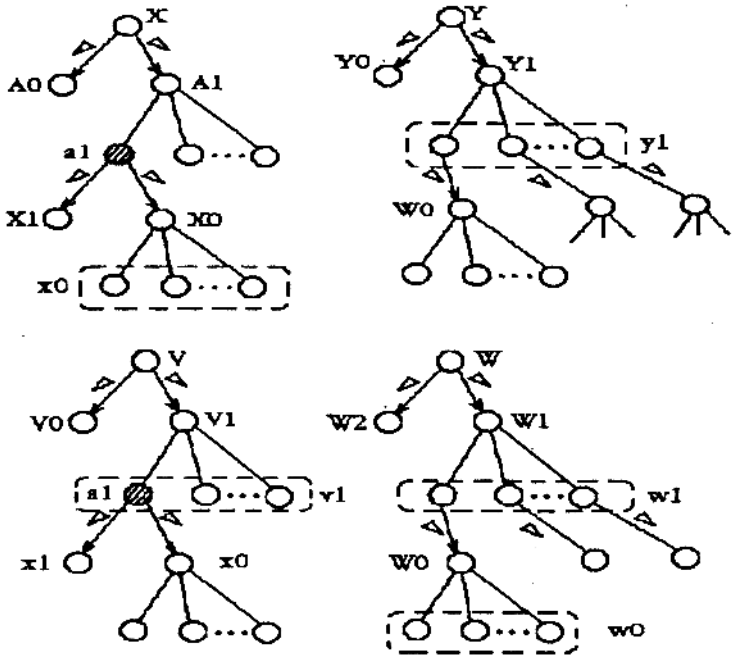


Рис. 2. Иерархические структуры проблемной области.

8. Полагаем, что W$\bigcap$Y=Z ($\neq\oslash$) и $W_1\bigcap Y_1$=Z_1 ($\neq\oslash$).

9. S_3=$S_2\circ${z/y_1, z/w_1} W_3 = { P((a_1/A_1)//X, (g(((z/Z_1)//Z)/X_0)//a_1, (z/Z_1)//Z, (f((g(((z/Z_1)//Z)/X_0)//a_1)/W_0)//z), P((a_1/A_1)//X, (g(((z/Z_1)//Z) /X_0)//a_1, (z/Z_1)//Z, (w_0/W_0)//z) }

10. { (f((g(((z/Z_1)//Z)/X_0)//a_1)/W_0)//z, (w_0/W_0)//z } - множество рас-

согласований

11. S_4=S_3 ({ f((g((z/Z_1)//Z)/X_0)//a_1)/w_0 } W_4 = { P((a_1/A_1)//X, (g((z/Z_1)//Z)/X_1)//a_1, (z/Z_1)//Z, (f((g((z/Z_1)//Z)/X_0)//a_1)/W_0)//z), P((a_1/A_1)//X, (g((z/Z_1)//Z)/X_0)//a_1, (z/Z_1)//Z, (f((g((z/Z_1)//Z)/X_0)//a_1)/ W_0)//z) } = { P((a_1/A_1)//X, (g((z/Z_1)//Z)/X_0)//a_1, (z/Z_1)//Z, (f((g((z/Z_1) //Z) / X_0)//a_1)/W_0)//z) }

12. W_4 - одноэлементное множество; S=S_k стоп.

В качестве процедуры вывода предлагается использовать линейную входную резолюцию, которая является полной для хорновских дизъюнктов и обладает большой эффективностью.

3.3 Особенности использования линейной входной резолюции в многоуровневой логике

Рассмотрим особенности использования линейной входной резолюции в MLL с упорядоченными дизъюнктами. Следует отметить, что линейная входная резолюция полна для дизъюнктов Хорна. Связывание понятия упорядоченных дизъюнктов с линейной входной резолюцией не нарушает ее полноты, но существенно увеличивает эффективность метода.

Пусть S - конечное множество упорядоченных дизъюнктов Хорна, полученное после удаления кванторов $\forall$ и $\exists$.

S содержит следующие дизъюнкты (для записи которых используется прологовская нотация):

$A_1 ... A_m$

$A_{m+1} \leftarrow B_{1,1} \& ... \& B_{1,n1}$

...

$A_{m+l} \leftarrow B_{l,1} \& ... \& B_{l,nl}$

$\leftarrow B_{l+1,1} \& ... \& B_{nl+1}$

...

$\leftarrow B_{l+k,1} \& ... \& B_{l+k,nl+k}$, где

m - множество всех положительных упорядоченных дизъюнктов Хорна из S (фактов);

l - множество всех смешанных упорядоченных дизъюнктов Хорна из S (фактов);

k - множество всех отрицательных упорядоченных дизъюнктов Хорна из S (фактов).

Рассмотрим одно из применений дедуктивного вывода, а именно, получение экстенсионалов отношений, которое позволяет значительно сократить экстенсиональную составляющую БЗ. Для этого случая в качестве верхнего центрального дизъюнкта возьмем отрицание предикатной литеры, имя которой соответствует имени отношения, экстенсионалы которого получаем, поскольку принцип резолюции является процедурой опровержения:

$\neg C_j$, j = 1 ÷ m+l. Для центрального дизъюнкта возможны два случая:

1). C_j (j = 1 ÷ m+l) является A_i, i = 1 ÷ m.

2). C_j (j = 1 ÷ m+l) является A_{m+i}, i = 1 ÷ l.

Из всех известных модификаций линейной входной резолюции возьмем наиболее эффективную - метод поиска в глубину.

Для первого случая, используя метод поиска в глубину, получаем набор экстенсионалов отношения, имя которого соответствует центральному дизъюнкту.

Во втором случае метод поиска в глубину повторяем для каждой посылки $B_{i,k}$, k = 1 ÷ n_i, входящей в логическое предложение, заключением которого является A_{m+i}, i = 1 ÷ l.

Особенностями линейной входной резолюции в MLL являются :

1) возможные значения термов (их домены) при резольвировании определяются по иерархической структуре;

2) существенное сокращение пространства поиска, которое влечет увеличение эффективности дедуктивного вывода, которое достигается за счет особенности 1.

Приведем в качестве примера использование дедуктивного вывода для получения экстенсионалов отношения

поступают(у, принтер),

которое определяет результирующие данные программных компонент некоторой функциональной системы #S, поступающие на принтер.

Запись в MLL:

(∃x//#S)(∃(y/Результат)//x) Поступают(y,принтер)

Поскольку принцип резолюции основан на процедуре опровержения, отрицание вышеприведенной формулы используется в качестве верхнего центрального дизъюнкта:

¬[∃(x//#S)(∃(y/Результат)//x) Поступают(y,принтер)] =

(∀x//#S)(∀(y/Результат)//x) (Поступают(y,принтер) (4)

Применяя алгоритм Сколемизации к формуле (4), получим:

¬Поступают((y/Результат)//x, принтер) (5)

Если запрос касается описания некоторой вершины и описания ее нижележащих вершин по иерархической структуре, как в рассматриваемом примере, мы выводим не пустой дизъюнкт, а предикат ответа. И формула (5), которая выступает в качестве центрального дизъюнкта, имеет вид :

¬Поступают((y/Результат)//x, принтер) ∨
ANSWER(x//#S,(y/Результат)//x)

Рассматриваемый пример представляет запрос 1-го вида. В случае, если запрос касается описания только самих вершин и не касается описаний их нижележащих вершин по иерархической структуре, то при получении ответа на запрос выводится пустой дизъюнкт и выдается список значений переменных, означивание которых произошло в процессе вывода. Такие запросы являются запросами 2-го вида.

В качестве примера запроса 2-го вида рассмотрим запрос: "Найти все программные компоненты функциональной системы #S, такие что объем оперативной памяти занимаемый ими, не превосходит 200 кб", который п редставляется формулой:

(∀x//#S) (∀y/REAL) Объем_памяти(x,y) & LE(y,200)

Запрос 2-го вида является частным случаем запроса 1-го вида, который выделяется, поскольку большинство запросов имеют такую форму.

Пусть в интенсиональной составляющей БЗ хранится множество аксиом, одной из которых является:

(∃x//#S)(∀(y/Результат)//x) Соединена_ЭВМ(x,принтер) & Время_получения(y,0.01c) → Поступают(y,принтер) (6)

и в экстенсиональной составляющей БЗ хранится множество фактов, одними из которых являются

Соединена_ЭВМ(ПК$_i$**,принтер), i = 1÷N и**
Время_получения(P_j**,0.01c), j = 1÷K.**

Применяя алгоритм Сколемизации к формуле (6), получим:

Соединена_ЭВМ(@a//#S,принтер) &
Время_получения((y/Результат)//@a,0.01c) →
Поступают((y/Результат)//@a,принтер),

где **@a**- обобщенная константа.

Цепочка вывода в виде дерева вывода представлена на рис. 3.

Множество Пк$_i$, i=1÷N, которые входят в состав функциональной системы #S, определяем по иерархической структуре, и для каждого ¡находим факт **Соединена_ЭВМ**(ПК$_i$,**принтер**).По иерархической структуре определяем множество P_j, j = 1÷K, которые являются результатами ПК$_i$.Для каждой пары { ПК$_i$, P_j } находим факт **Время_получения**(P_j, **0.01с**).Ответом на запрос будет множество пар { ПК$_i$,P_j}, являющихся значениями переменных x и y соответственно.

Разработанные алгоритмы дедуктивного вывода для предложенного расширения синтаксиса MLL положены в основу системы моделирования сложноструктурированной проблемной области KM (Knowledge Model).

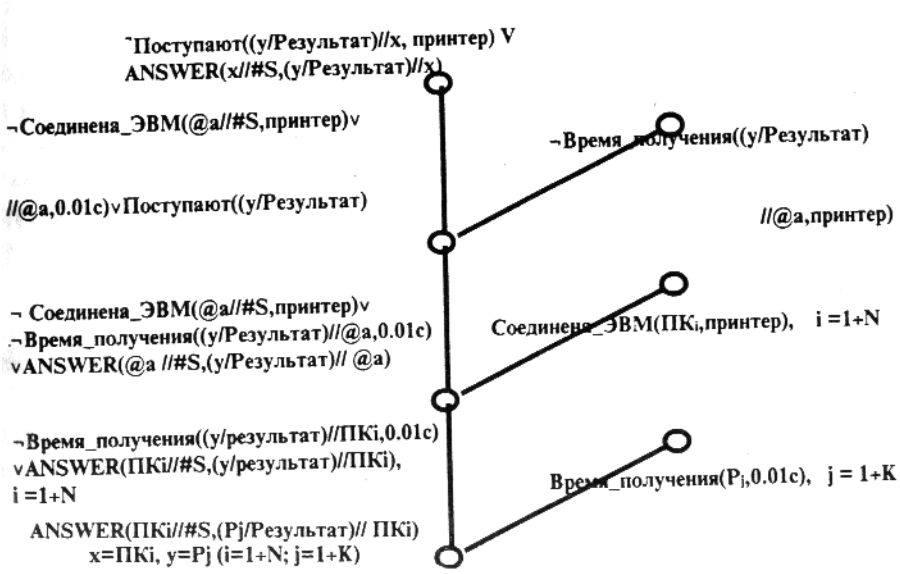


Рис. 3. Дерево вывода.

4 Иерархическая структура и продукционная модель

Поскольку проблемная область большинства проектируемых программных комплексов в CASE-системе является сложноструктурированной и динамической, и она представляется в базе знаний репозитория, рассмотрим средства для ее формализации. Из вышерассмотренного следует, что MLL является удобным средством для формализации структурного аспекта проблемной области.

Иерархическую абстрактную структуру удобно использовать и при описании проблемной области продукционной моделью представления знаний, которая применяется для отражения динамики изменения проблемной области, когда для ее задания требуется огромное количество (1-10 тыс.) продукционных правил.

Иерархическая абстрактная структура позволяет разбить продукционные правила на блоки в соответствии с принадлежностью к элементам структуры и использовать механизм наследования продукционных правил. Механизм наследования продукционных правил позволяет "сжать"базу знаний, сделать ее более компактной. Рассмотрим механизм наследования продукционных правил на примере. Пусть задана иерархическая структура, которая описывает некоторую абстрактную проблемную область "Аэропорт". Иерархическая структура состоит и 3-ех уровней детализации:

1-й уровень - Аэропорт;

2-й уровень - РЛС, классы самолетов (ТУ, ИЛ, АН,...);

3-й уровень - представители классов РЛС, самолетов (конкретные объекты).

Управление работой аэропорта задается множеством продукционных правил, которые разбиваются на блоки в соответствии с уровнями в иерархической структуре следующим образом:

- в блок 1-го уровня входят продукционные правила, которые описывают принципы управления аэропортом в целом;

- в блоки 2-го уровня входят продукционные правила, которые описывают управление РЛС и классами самолетов;

- число блоков 3-го уровня определяется количеством представителей классов РЛС и самолетов.

В каждый блок 3-го уровня входят продукционные правила, которые задают специфические законы управления конкретным самолетом или РЛС. А остальные продукционные правила, которые задают общие законы управления РЛС и классами самолетов, могут быть получены благодаря механизму наследования.

Таким образом, иерархическая структура позволяет создать иерархию продукционных правил и использовать принцип наследования продукционных правил, подобно механизму наследования свойств в ISA иерархии.

Список литературы

[1] Modern Software Engineering. Foundation and Current Perspectives.- Edited by Peter A.Ng., Raymond T. Yeh. - VAN NOSTRAND REINHOLD, New York,-1990.- c.591.

[2] Cooling. Software Design for Real-time Systems. - CHAPMAN AND HALL (University and Professional Division).- 1991, p. 505

[3] Harel D. and ets. Statemate: a Working Environment for the Development of Complex Reactive Systems.-IEEE Transactions on Software Engineering, Vol. 16, No. 4, April 1990, pp.403-413

[4] Robert V. Rubin, James Walker II, Eric J. Golin Early Experience with the Visual Programmer's Workbench.- IEEE Transactions on Software Engineering, vol.16, No.10,1990

[5] Andrew J.Symohds Creating a Software-Engineering Knowledge Base.- Software Development Computer-Aided Software engineerig (CASE).Edited by Chirofsky E.J.- IEEE Computer Society press technology series, 1989

[6] Marc Eisenstadt, J.Domingue, T.Rajan, E.Motta Visual Knowledge Engineering,- IEEE Transactions on Software Engineering, Vol.16, No. 10, October 1990

[7] Eisenstadt, M. Brayshaw A Knowledge Engineering Toolkit.- BYTE, vol.15, No.10,12,1990

[8] Вагин В.Н., Головина Е.Ю., Салапина Н.О. Искусственный интеллект в CASE-технологии. Программные системы и продукты -НИИ "Центрпрограммсистем".Тверь.-N 3, 1996., с.13-21

[9] Sally Shlaer and Stephen J. Mellor "Object Lifecycles: Modeling the World in States",Prentice-Hall, Englewood Cliffs, N.J., 1992

[10] Gibson Objects - Born and Bred.- BYTE, October,1990

[11] Yokoyamat An object-oriented and constraint-based knowledge representation system for design object modeling.- Tokyo, Japan, ICOT Research Center,1989

[12] Ohsuga S., Yamauchi H. Multi-layer logic - a predicate logic including data structure as knowledge representation language.- New generation computing, Vol.3,-NO.4,-1985 -c.451-485.

[13] Ohsuga S. Toward inlelligent CAD systems.- Computer Aided Desing, Vol.21,-NO.5,- 1989.-c.315- 337.