УДК 621.865.8

С. А. НЕСТЕРЦОВА, О. В. ПОПОВА, А. М. ЦЫМБАЛ

РЕАЛИЗАЦИЯ СИСТЕМЫ РАЗРАБОТКИ ПЛАНОВ ДЛЯ ИНТЕЛЛЕКТУАЛЬНОГО РОБОТА НА ЯЗЫКЕ С++

Система управления интеллектуальным роботом должна обладать механизмами разрешения задач, возникающих в результате деятельности такого робота. При этом сложность реализации такого решателя задач естественно зависит от сложности предметного пространства, в котором интеллектуальный робот функционирует. Однако существуют и общие для всех такого рода программ особенности, составляющие суть исследований в данной области искусственного интеллекта.

В литературе хорошо описана схема интеллектуального решателя STRIPS, разработанного еще в начале 70-х. Проблемная среда STRIPS включает модель мира, набор операторных схем и цель. Модель мира представляется в виде набора правильно построенных формул (ППФ) логики предикатов первого порядка, которые выражают собой факты и законы. Операторная схема определяется наименованием, списком параметров и описаниями в виде ППФ языка логики предикатов первого порядка: условий применения действия и результата действия. Результата действия содержит список вычеркиваний и список добавлений. Операторы порождают различные модели мира путем генерирования новых фактов. Цель также представляется в виде ППФ той же логики [1,2].

Пример реализации системы разработки планов на Turbo-Prolog приведен в [3]. Описанная система решала задачу принятия решений интеллектуальным роботом в замкнутом предметном пространстве при ограниченном наборе связанных с пространством событий.

В силу распространенности и универсальности такого языка как С++, разумно рассмотреть реализацию системы разработки планов с позиций С++.

Прежде всего, следует говорить о разработке класса, который обеспечивал бы всю работу программы с базой данных. Работа с базой данных являлась и основной особенностью программы на Turbo-Prolog. В класс включаются компоненты, несущие информацию об именах отношений, описывающих базу данных, действующий объект отношения и, соответственно, субъекты применения действия объекта.

Как и в [1], рассматривается замкнутое пространство объектов и событий, состоящее из следующего набора: комнаты; двери, соединяющие комнаты; ящики в комнатах; робот, перемещающийся из комнаты в комнату и осуществляющий действия.

Например, факт наличия ящика box1 в комнате room1 будет описываться как:

dbfact("is_in","box","room1",NULL,aN),

где «is_in» - имя отношения, «box» - объект; room1 – имя субъекта; NULL – указатель на отсутствующий (для данного случая) субъект, aN – указатель на следующий в цепочке описания факт базы данных (база данных реализована связанным списком). В соответствии с описанием факта в классе реализованы соответствующие конструкторы.

Общий вид класса можно описать следующим образом:

```
class dbfact {char *relname, *agent, *sobj1, *sobj2;
              dbfact *next;
              public:
                    dbfact(....); // перегружен
                    void print();
                    dbfact *dataformer();
                    dbfact *addfact(....); // перегружен
                    int dbtest(....);// перегружен
                    dbfact *delfact(....);  // перегружен
                    dbfact *findfact(....); // перегружен
                    void deciser();
```

```
char *findaction();
void support_action(char *);
void old_cleaner(char *);
void new_adder(char *);
};
```

Таким образом в одном классе реализованы все функции работы с базой данных программы. Введение класса упрощает многие процедуры обработки данных

Для упрощения доступа, база данных объявлена глобальной в виде указателя evil на связанный список, реализующий хранение данных.

Теперь об описании собственно программы. Функция main() описана следующим образом:

```
void main()
{dbfact *goal; // объявление переменной целевого факта
 evil = evil -> dataformer(); // заполнения базы данных программы
 goal= new dbfact("is_at", "robot", "door23",NULL);
   // пример постановки задачи в программе
 goal-> deciser(); // вызов механизма принятия решений для
   // поставленной задачи
}
```

Таким образом, для сформированного целевого файла будеи вызвана процедура генерации планов.

Функция нахождения решения реализована так:

```
void dbfact::deciser()
{if(!evil->dbtest(relname,agent,sobj1,sobj2))
 {char *action=find_action();
 support_action(action);
 cout << «\n\n Action : « << action;getch();
 old_cleaner(action);
 new_adder(action);
 }}
```

Работает функция deciser() следующим образом. Первоначально осуществляется проверка – существует ли уже в базе данных принятый к рассмотрению факт – проверка dbtest(…). Если таковой существует в базе, в процессе нахождения решений нет необходимости и мы возвращаемся в точку вызова решателя. Если же факта нет, необходимо найти такое действие, которое могло бы потенциально такой факт обеспечить. С этой целью запускается функция find_action(), которая должна найти необходимое действие. В принципе, одного нахождения имени действия явно мало, и поэтому следующий шаг – вызов support_action(action) собственно решает задачу реальной реализации действия action. Две следующие функции old_cleaner(action) и new_adder(action) обеспечивают необходимую обработку базы данных программы.

Нахождение необходимого имени действия реализовано следующим образом. Например, поставлена цель – «открыть дверь door 12». Здесь необходимы следующие проверки: во-первых, проверить, действительно ли объект door12 дверь (т.е. может ли быть к нему применена операция «открыть») и во-вторых - проверить факт, альтернативный постановке задачи, (т.е. закрытое состояние двери). При позитивном результате таких проверок возвращается имя действия.

Пример реализации (для действия «открыть дверь»):

```
char *dbfact::find_action()
{if(evil->dbtest(«is_a»,agent,»door») &&
   evil->dbtest(«stands»,agent,»closed»))
   return «open_door»;
 // описания для других действий
 else return «No action expected»;
}
```

Когда имя целевого (т.е., необходимого для обеспечения поставленной цели) действия найдено, следует обеспечить поддержку решения. Например, чтобы закрыть (открыть) дверь, нужно как минимум, возле нее находиться.

Функция support_action(char) обеспечивает необходимый поддерживающий набор действий.

Пример реализации:

```
void dbfact::support_action(char *action)
{ if(!(strcmp(action, «open_door»)))
  {dbfact *subgoal=new dbfact(«is_at», «robot»,agent,NULL);
   subgoal->deciser();}
 ……..
}
```

В функцию передается имя действия. Каждому действию соответствует свой «поддерживающий набор».

Если имя такого набора и переданное в функцию имя действия совпадают, что обеспечивает функция проверки совпадения строк, объявляется подцель subgoal, специфичная для каждого набора поддержки. После этого, механизм принятия решений вновь (и теперь уже рекурсивно) вызывается для подцели (подцелей). Он будет следовать аналогичной рассматриваемой схеме. Как только весь поддерживающий набор выполнен, можно считать, что действие полностью "подготовлено" и следует лишь удалить устаревшие по завершению действия факты и добавить набор фактов, характеризующий возникшую по завершению действия новую предметную ситуацию. Это событие характеризуется функциями new_adder(…) и old_cleaner(…).

Пример реализации:

```
void dbfact::old_cleaner(char *action)
{if(!(strcmp(action,»open_door»)))
  {evil->delfact(«stands»,agent,»closed»,NULL);}
  ……..
}
void dbfact::new_adder(char *action)
{ if(!(strcmp(action,»open_door»)))
  {evil=evil->addfact(«stands»,agent,»opened»,NULL);}
 ……..
}
```

Собственно, эти функции распределяют, в соответствие с каким действием какие факты необходимо соответственно удалить и добавить, а реальное удаление и добавление осуществляется функциями del_fact(…) и add_fact(…), которые выполняют нужную работу со связанным списком базы данных.

Среди реализованных в системе действий описаны действия робота: закрыть дверь, открыть дверь, перейти к объекту, толкать объект, перейти в комнату.

Развитием описанной программы является ее реализация в графической среде программирования. Планируется расширить круг решаемых задач, разработать более эффективные схемы нахождения решений.

Материалы статьи использованы авторами при проведении занятий для студентов ХТУРЭ 4-го курса направления «Компьютеризированные системы управления и автоматики».

Список литературы: 1. *Ефимов Е.И.* Решатели интеллектуальных задач М.: Наука, 1982. 320 с. 2. *Нильсон Н.* Искусственный интеллект. М.: Мир, 1984. 362 с. 3. *Цымбал А.М., Попова О.В.* Реализация системы разработки планов на Turbo-Prolog // Авиационно-космическая техника и технология. Вып. 18. Харьков. Государственный аэрокосмический университет «ХАИ». 2000. С. 68-71.

Поступила в редколлегию 7.07.2000