Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

И. В. Марьясов

ПРИМЕНЕНИЕ СМЕШАННОЙ АКСИОМАТИЧЕСКОЙ СЕМАНТИКИ ЯЗЫКА C-KERNEL К ВЕРИФИКАЦИИ ПРОГРАММЫ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ

Препринт
155

Новосибирск 2010

В настоящей работе проводится верификация методом Хоара программы топологической сортировки с использованием смешанной аксиоматической семантики языка C-kernel, являющегося ядром языка C-light.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

I. V. Maryasov

APPLYING THE MIXED AXIOMATIC SEMANTICS OF C-KERNEL LANGUAGE TO THE VERIFICATION OF TOPOLOGICAL SORTING PROGRAM

Preprint
155

Novosibirsk 2010

A topological sorting program was verified by applying mixed modified axiomatic semantics of C-kernel language. This language is the kernel of C-light language.

1. ВВЕДЕНИЕ

Данный препринт продолжает цикл работ по верификации программ на языке C-light [1], являющимся представительным подмножеством языка C. Во избежание громоздкости аксиоматической семантики в языке C-light было выделено ядро – язык C-kernel, для которого была разработана аксиоматическая семантика, и в который транслируются исходные программы на языке C-light. По сравнению с языком C-light в нём более простые выражения и более ограниченный набор операторов. Однако предложенная семантика не ориентирована на автоматизацию процесса вывода условий корректности, поскольку она сконструирована для отдельных операторов независимо от контекста. Кроме того, условия корректности даже для небольших программ получались громоздкими, что затрудняло их последующее доказательство.

В [2] была предложена смешанная аксиоматическая семантика языка C-kernel. Термин «смешанная» означает, что в ней имеются правила вывода специального вида, применяемые, когда в анализируемом фрагменте программы нет переменных, доступ к значению которых осуществляется через указатели. Во многих случаях это позволяет существенно упростить условия корректности. Дальнейшая работа над семантикой [8] позволила упростить используемый язык утверждений, правила вывода и, как следствие, выводимые условия корректности. В настоящей работе проводится верификация программы топологической сортировки с использованием разработанной смешанной аксиоматической семантики.

Работа имеет следующую структуру. В разделе 2 описаны изменения по сравнению с [2] языка утверждений, используемого для описания свойств программы. Раздел 3 даёт обзор смешанной аксиоматической семантики MHSC языка C-kernel. В разделе 4 проводится верификация программы топологической сортировки. Полный список правил вывода MHSC дан в приложении 1, тексты программы топологической сортировки на языках C-light и C-kernel приведены в приложении 2.

2. ЯЗЫК УТВЕРЖДЕНИЙ

В язык утверждений по сравнению с [2] были внесены следующие изменения:

1. Уменьшено количество метапеременных: исключены метапеременные MeM и TP.

2. Элиминированы lv-типы: вместо них введена абстрактная функция addr и изменено определение функции val.

3. Изменены определения некоторых абстрактных функций.

В [2] для определения адреса объекта в памяти по его идентификатору использовалось отображение MeM: ID → Locations, где ID – множество допустимых идентификаторов языка C-light, Locations – множество адресов ячеек памяти. Однако во время выполнения программы адрес сам по себе не изменяется, может изменяться лишь значение в ячейке памяти по данному адресу. Таким образом, с учётом требования уникальности идентификаторов языка C-kernel каждой программной переменной можно сразу же сопоставить адрес, по которому хранится её значение в памяти: если переменная имеет идентификатор x, то через &x обозначим адрес – элемент множества Locations.

Аналогично замечаем, что тип объекта во время выполнения программы также не меняется, следовательно, тип содержимого памяти по конкретному адресу можно вычислить один раз. Тем самым, мы избавляемся от отображения TP: Locations → LogTypeSpecs, где LogTypeSpecs – множество логических имён типов, которые являются логическими представлениями абстрактных имён типов языка C-light. Вычисление типа осуществляет *семейство функций type*: если существует декларация τ x;, то type(x) = τ, в остальном определение из [2] сохраняется.

Таким образом, в определениях всех абстрактных функций исчезает зависимость от аргументов MeM и TP. Символом ω обозначим неопределённость значения.

Функция addr(e, MD) вычисляет адрес выражения e в соответствии со значением метапеременной MD:

- addr(e, MD) = &e, если e – обычная переменная;
- addr(e, MD) = ω, если e – константа или переменная, к значению которой нет обращения через указатели;
- addr(e[e'], MD) = mb(e, val(e', MD, STD));
- addr(e[e'], MD) = ω, если e – массив, к значениям элементов которого нет обращения через указатели;
- addr(e.m, MD) = mb(val(e, MD, STD), val(m, MD, STD));
- addr(e.m, MD) = ω, если e – структура, к значениям полей которой нет обращения через указатели;
- addr(&e, MD) = ω;
- addr(*e, MD) = val(e, MD, STD).

Функция val(e, MD, STD) вычисляет значение выражения e в соответствии со значениями метапеременных MD и STD:

- val(e, MD, STD) = MD(&e), если e – обычная переменная;
- val(e, MD, STD) = e, если e – константа или переменная, к значению которой нет обращения через указатели;
- val(e[e'], MD, STD) = MD(mb(e, val(e', MD, STD)));
- val(e[e'], MD, STD) = e[val(e', MD, STD)], если e – массив, к значениям элементов которого нет обращения через указатели;
- val(e.m, MD, STD) = MD(mb(val(e, MD, STD), val(m, MD, STD)));
- val(e.m, MD, STD) = e.m, если e – структура, к значениям полей которой нет обращения через указатели;
- val(&e, MD, STD) = addr(e, MD);
- val(*e, MD, STD) = MD(val(e, MD, STD));
- val((τ) e, MD, STD) = cast(val(e, MD, STD), type(e), fst(logtype(τ, MD, STD)));
- val(•e, MD, STD) = UnOpSem(•, val(e, MD, STD), type(e)), где • – унарная операция;
- val(e • e', MD, STD) = BinOpSem(•, val(e, MD, STD), type(e), val(e', MD, STD), type(e')), где • – бинарная операция.

3. СМЕШАННАЯ АКСИОМАТИЧЕСКАЯ СЕМАНТИКА MHSC ЯЗЫКА C-KERNEL

Изложенные в разд. 2 изменения привели к изменениям в правилах вывода системы MHSC, причём не только формальным, связанным с изменением в определениях абстрактных функций, участвующих в этих правилах, но и к структурным: в связи с элиминацией метапеременной MeM сократилось число правил вывода для блока, а также упростились сами правила. Теперь они имеют следующий вид:

$$\frac{(f, \tau, B, id, E_5), SP \vdash \{P\}\ S;\ A;\ \{Q\}}{(f, \tau, B, cur, E_5), SP \vdash \{P\}\{S\}_{id};\ A;\ \{Q\}} \quad (1)$$

$$\frac{(f, \tau, B, cur, E_5), SP \vdash \{P\}\ A;\ \{Q\}}{(f, \tau, B, id, E_5), SP \vdash \{P\}\ blockEnd(cur);\ A;\ \{Q\}}\ id \neq E_5 \quad (2)$$

$$\frac{SP_{fun} \models P \Rightarrow SP_{post}(f)}{(f, \tau, B, id(f), id(f)), SP \vdash \{P\}\ blockEnd(cur);\ A;\ \{Q\}} \quad (3)$$

Правило (1) раскрывает скобки блока, помечая его конец специальной конструкцией blockEnd, которая элиминируется правилами (2) и (3).

Правило (2) соответствует случаю обычного выхода из блока и обеспечивает просачивание метки L (если был встречен оператор goto L, но не найден оператор, помеченный этой меткой L) и идентификатора тела функции f (если не был встречен оператор return).

В случае правила (3) мы нашли соответствующую закрывающую скобку для ранее встреченного оператора return: обработка тела функции завершена, порождается условие корректности.

Полный список правил приведён в приложении.

4. ВЕРИФИКАЦИЯ ПРОГРАММЫ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ

Текст программы топологической сортировки был взят из [3] и переписан на языке C-light. Напомним, что топологическая сортировка есть преобразование частичного порядка, заданного на некотором конечном множестве, в линейный. При этом предполагается, что этот порядок задан конечным ациклическим орграфом.

Идея алгоритма заключается в следующем: на первом шаге выбираем любую вершину, у которой нет предшественников (она всегда есть, т. к. в противном случае в графе будет цикл, что противоречит условию задачи) и помещаем её в голову результирующего списка. Удаляем эту вершину в исходном графе вместе со всеми дугами, исходящими из неё. Получится граф, также удовлетворяющий условиям задачи. Продолжаем данный процесс до тех пор, пока граф не станет пустым.

Для представления дуг исходного графа используется структура edge, состоящая их двух полей p и q, которые представляют номера вершин, которые соединяет дуга. Сам граф задан в виде массива edges структур edge (в оригинальной программе данная информация считывается из файла).

Для сортировки в программе используется специальное внутреннее представление графа, основанное списках структур leader и trailer. Список из структур leader называется списком ведущих, каждый его элемент содержит следующую информацию:

– поле key хранит номер вершины графа;

– поле count содержит количество вершин-предшественников вершины с номером key;
– поле trail – указатель на список ведомых;
– поле next – указатель на следующий элемент списка ведущих.

Список ведомых, состоящий из структур trailer, используется для хранения информации о дугах, исходящих из вершины key. Каждый его элемент имеет следующий вид:

– поле id – указатель на элемент в списке ведущих, т. е. на ту вершину, в которую идёт дуга из вершины key в графе;
– поле next – указатель на следующий элемент списка ведомых.

На первом этапе происходит перевод информации о графе из массива edges во внутреннее представление, основанное на списке ведущих. Переменная head указывает на его первый элемент, а tail – на последний. Данный перевод осуществляет функция find, которая возвращает указатель на элемент списка ведущих, если данная вершина уже была считана раньше, либо в противном случае добавляет её в конец списка, также возвращая указатель.

Далее проводится оптимизация: в полученном списке ведущих расставляются связи между вершинами без предшественников, т. е. элементами, у которых поле count равно нулю.

После этого начинает работу главный цикл, который проходит по всем вершинам преобразованного списка ведущих. Внутренний цикл идёт по списку ведомых, при этом каждый раз, когда получается вершина без предшественников, она добавляется в список ведущих.

Результат заносится в список, на который указывает переменная res: он хранит вершины в порядке топологической сортировки.

Тексты программы топологической сортировки на языках C-light и C-kernel приведены в прил. 2.

С целью спецификации программы введём вспомогательные предикаты и функции.

Функция tuple(x, y) возвращает кортеж значений элементов списка, начиная с элемента x и y.

Функция dellast($(n_1, \ldots, n_{k-1}, n_k)$) удаляет последний элемент кортежа:

$$\mathrm{dellast}((n_1, \ldots, n_{k-1}, n_k)) = (n_1, \ldots, n_{k-1}).$$

Пусть $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_k)$, тогда *функция con(X, Y)* осуществляет соединение элементов кортежей X и Y:

$$con(X, Y) = (x_1, ..., x_n, y_1, ..., y_k).$$

Из определений этих двух функций немедленно следует свойство:

$$tuple(x, y) = con(dellast(tuple(x, y)), tuple(y, y)).$$

Предикат reach(y, x) истинен тогда и только тогда, когда элемент списка $y \neq x$ достижим из элемента x, то есть существует конечная последовательность x -> next -> ... -> next = y, а reach(x, x) – тождественная истина.

Непосредственно из определения получаем:

$$reach(y, x) \wedge reach(z, y) \Rightarrow reach(z, x).$$

Специфицируем функцию find:

- предусловие $SP_{pre}(find) \equiv w = w_0$;
- инвариант $INV(find) \equiv w = w_0 \wedge w_0 \notin dellast(tuple(MD(\&head), MD(\&h))) \wedge reach(MD(\&h), \&head)$;
- постусловие $SP_{post}(find) \equiv w = w_0 \wedge w_0 \notin dellast(tuple(MD(\&head), MD(\&fst(Val)))) \wedge reach(fst(Val), \&head) \wedge MD(mb(fst(Val), key)) = w_0$.

Предусловие $SP_{pre}(find)$ фиксирует начальные данные – целое число w_0, которое нужно найти в списке. Инвариант цикла INV(find) утверждает, что искомое число w_0 не встретилось в уже просмотренной части списка, а текущий элемент h достижим из начала списка head. Постусловие $SP_{post}(find)$ выражает тот факт, что искомое число w_0 было найдено в списке и функция find возвращает указатель на этот элемент.

В соответствии с правилом вывода для всей программы (П.36) вывод условий корректности проводится для каждой функции, присутствующей в тексте программы. Выведем условия корректности для функции find.

В силу правила вывода для оператора цикла (П.28) первое условие корректности соответствует участку тела функции find от её начала до цикла. Мы имеем декларацию переменной h и две операции присваивания. Применяя последовательно соответствующие правила, получим:

$\exists MD_4$ (
  $\exists MD_3$ (
    $\exists MD_1\ \exists STD_1\ \exists nc\ \exists \tau\ \exists V\ \exists MD_2\ \exists STD_2$ (
      $SP_{pre}(find)(w) \wedge$
      $(LPtr, STD_2) \in (pointer(struct(leader)), STD_1) \wedge$

```
      MD1(nc) = ⊥ ∧
      (MD2, V) ∈ init(LPtr, auto, upd(MD1, nc, ω)) ∧
      MD3 = upd(MD2, nc, fst(V)) ∧
      STD = STD2
    ) ∧
    MD4 = upd(MD3, &h, MD3(&head))
  ) ∧
  MD = upd(MD4, mb(MD(&tail), key), MD(&w))
) ⇒
INV(find)(w, MD).
```

Докажем данное условие корректности. В силу того, что $MD_4 = upd(MD_3, \&h, MD_3(\&head))$, получаем MD(&head) = MD(&h). Следовательно, tuple(MD(&head), MD(&h)) = tuple(MD(&head), MD(&head)) – одноэлементный кортеж, а значит, функция dellast вернёт пустой кортеж, и второй конъюнкт будет истинен. Третий же конъюнкт тождественно истинен по определению предиката reach. Таким образом, данное условие истинно.

Второе условие корректности согласно правилу вывода для цикла получится при рассмотрении тройки Хоара {INV(find) ∧ α} A {INV(find)}, где α – условие цикла, A – его тело. Но оно состоит из единственного оператора присваивания. Поэтому применяя правило (П.5), сразу же получаем следующее условие корректности:

```
∃MD' (
  INV(find)(w, MD') ∧
  MD'(mb(MD'(&h), key)) ≠ w ∧
  MD = upd(MD', &h, MD'(mb(MD'(&h), next)))
) ⇒
INV(find)(w, MD).
```

В силу равенства MD = upd(MD', &h, MD'(mb(MD'(&h), next))) получаем, что MD(&h) = MD'(mb(MD'(&h), next)). Тогда reach(MD(&h), &head) = reach(MD'(mb(MD'(&h), next)), &head). Но в посылке мы имеем reach(MD'(&h), &head), кроме того, по определению истинен предикат reach(MD'(mb(MD'(&h), next)), MD'(&h)). Воспользовавшись свойством предиката reach, получаем требуемое.

tuple(MD(&head), MD(&h)) = tuple(MD(&head), MD'(mb(MD'(&h), next))), но тогда dellast(tuple(MD(&head), MD'(mb(MD'(&h), next)))) = tuple(MD'(&head), MD'(&h)). В силу того, что $w_0 \notin$ dellast(tuple(MD'(&head), MD'(&h))) и MD'(mb(MD'(&h), key)) ≠ w получаем истинность второго конъюнкта заключения.

Тем самым данное условие корректности доказано.

В оставшейся части текста функции find (после цикла) имеется условный оператор, поэтому в силу правила (П.27) получим ещё два условия корректности.

$$
\begin{array}{l}
\exists Val'\ (\\
\quad \exists MD_5\ (\\
\quad\quad \exists MD_4\ (\\
\quad\quad\quad \exists MD_3\ (\\
\quad\quad\quad\quad \exists MD_1\ \exists STD_1\ \exists MD_2\ \exists STD_2\ \exists \tau\ \exists V\ (\\
\quad\quad\quad\quad\quad INV(find)(w, MD_1) \wedge\\
\quad\quad\quad\quad\quad MD_1(mb(MD_1(\&h), key)) = w \wedge\\
\quad\quad\quad\quad\quad MD_1(\&h) = MD_1(\&tail) \wedge\\
\quad\quad\quad\quad\quad (\tau, STD_2) \in logtype(pointer(struct(leader)), MD_3, STD) \wedge\\
\quad\quad\quad\quad\quad (MD_2, V) \in new(\tau, MD_3) \wedge\\
\quad\quad\quad\quad\quad MD_3 = upd(MD_2, \&tail, cast(fst(V), snd(V), pointer(struct(leader)))) \wedge\\
\quad\quad\quad\quad\quad STD = STD_2\\
\quad\quad\quad\quad)\ \wedge\\
\quad\quad\quad\quad MD_4 = upd(MD_3, mb(MD_3(\&h), count), 0)\\
\quad\quad\quad)\ \wedge\\
\quad\quad\quad MD_5 = upd(MD_4, mb(MD_4(\&h), trail), NULL)\\
\quad\quad)\ \wedge\\
\quad\quad MD = upd(MD_5, mb(MD_5(\&h), next), MD_5(\&tail))\\
\quad)\ \wedge\\
\quad Val = (MD(\&h), pointer(struct(leader)))\\
) \Rightarrow\\
SP_{post}(find)(w, MD).
\end{array}
$$

Так как Val = (MD(&h), pointer(struct(leader))), то имеем fst(Val) = MD(&h). Последовательно подставляя, получаем, что MD(&head) = MD_1(&head), MD(&h) = MD_1(&h), следовательно, условие корректности истинно.

$$
\begin{array}{l}
\exists Val'\ (\\
\quad INV(find)(w, MD) \wedge\\
\quad MD(mb(MD(\&h), key)) = w \wedge\\
\quad MD(\&h) \neq MD(\&tail) \wedge\\
\quad Val = (MD(\&h), pointer(struct(leader)))\\
) \Rightarrow\\
SP_{post}(w, MD).
\end{array}
$$

В силу равенства Val = (MD(&h), pointer(struct(leader))) имеем fst(Val) = MD(&h), следовательно, данное условие корректности истинно.

Таким образом, функция find частично корректна.

Перейдём теперь к функции main. Спецификация имеет следующий вид:

- предусловие $SP_{pre}(main) \equiv \forall l\ \neg(\exists i_1, ..., i_l\ MD(mb(mb(edges, i_1), q)) = MD(mb(mb(edges, i_2), p)) \wedge MD(mb(mb(edges, i_2), q)) = MD(mb(mb(edges, i_3), p)) \wedge ... \wedge MD(mb(mb(edges, i_l), q)) = MD(mb(mb(edges, i_1), p)))$;
- инвариант $INV_1(main) \equiv \forall l\ \neg(\exists f_1, ..., f_l, g_1, ..., g_l\ f_1 \in tuple(MD(\&head), MD(\&tail)) \wedge ... \wedge f_l \in tuple(MD(\&head), MD(\&tail)) \wedge g_1 \in ttuple(MD(mb(f_1, trail))) \wedge ... \wedge g_l \in ttuple(MD(mb(f_l, trail))) \wedge MD(mb(MD(mb(g_1, id)), key)) = MD(mb(f_2, key)) \wedge ... \wedge MD(mb(MD(mb(g_l, id)), key)) = MD(mb(f_1, key)))$;
- инвариант $INV_2(main) \equiv (MD(\&head) = NULL) \vee (\forall f\ (f \in ttuple(MD(\&head)) \Rightarrow MD(mb(f, count)) = 0)) \wedge topsort(MD(\&res))$;
- инвариант $INV_3(main) \equiv (\forall f\ (f \in ttuple(MD(\&head)) \Rightarrow MD(mb(f, count)) = 0)) \wedge topsort(MD(\&res))$;
- инвариант $INV_4(main) \equiv (\forall f\ (f \in ttuple(MD(\&head)) \Rightarrow MD(mb(f, count)) = 0)) \wedge topsort(MD(\&res)) \wedge (predcount(MD(mb(MD(mb(MD(\&t), id)), key)), MD(\&res)) = MD(mb(MD(mb(MD(\&t), id)), count)) + 1)$;
- постусловие $SP_{post}(main) \equiv topsort(MD(\&res))$.

Предусловие $SP_{pre}(main)$ задаёт начальные условия, а именно – отсутствие циклов в исходном графе. В первом программном цикле происходит считывание данных о дугах графа из массива в специальную структуру, описанную выше. Таким образом, инвариант этого цикла $INV_1(main)$ утверждает то же самое, что и предусловие, но только для этой структуры.

Второй цикл проводит оптимизацию, расставляя связи между элементами списка ведущих, у которых нет предшественников (поле count = 0), что и выражает соответствующий инвариант $INV_2(main)$.

Третий цикл выполняет топологическую сортировку, итерируя по элементам списка ведущих без предшественников и помещая номера вершин графа в результирующий список в нужном порядке. Данный факт выражен в инварианте $INV_3(main)$. Четвёртый цикл является внутренним для третьего цикла, он ведёт итерацию по элементам списка ведомых.

Выведем и докажем условия корректности функции main.

В теле функции имеется четыре цикла, а также два условных оператора. В силу правила (П.28) в процессе вывода каждый цикл порождает одно условие корректности и даёт ветвление в дереве вывода. Правило (П.27) даёт две ветви.

Первое условие корректности получим на участке тела функции main от её начала до первого цикла.

$$
\begin{array}{l}
\exists i' \, (\\
\quad \exists MD_3 \, (\\
\qquad \exists MD_1 \, \exists STD_1 \, \exists MD_2 \, \exists STD_2 \, \exists \tau \, \exists V \; SP_{pre}(main)(MD) \wedge \\
\qquad (\tau, STD_2) \in pointer(struct(leader)) \wedge \\
\qquad (MD_2, V) \in new(\tau, MD_3) \wedge \\
\qquad MD_3 = upd(MD_2, \&head, cast(fst(V), snd(V), type(head))) \wedge \\
\qquad STD = STD_2 \\
\quad) \wedge \\
\quad MD = upd(MD_3, \&tail, MD_3(\&head)) \\
\quad) \wedge \\
\quad i = 0 \\
) \Rightarrow \\
INV_1(main)(MD).
\end{array}
$$

Из посылки получаем, что MD(&head) = MD(&tail), поэтому кортеж tuple(MD(&head), MD(&tail)) – одноэлементный, и при этом в силу посылки mb(MD(&head), id) = NULL, следовательно, заключение истинно.

В соответствии с (П.28) переходим к телу цикла. Получим второе условие корректности:

$$
\begin{array}{l}
\exists i_1 \, (\\
\quad \exists MD_8 \, (\\
\qquad \exists MD_7 \, (\\
\qquad\quad \exists MD_6 \, (\\
\qquad\qquad \exists MD_5 \, (\\
\qquad\qquad\quad \exists MD_3 \, \exists MD_4 \, \exists \tau \, \exists V \, (\\
\qquad\qquad\qquad \exists Val_2 \, \exists MD_2 \, (\\
\qquad\qquad\qquad\quad \exists Val_1 \, \exists MD_1 \, (\\
\qquad\qquad\qquad\qquad \exists y_1 \, (\\
\qquad\qquad\qquad\qquad\quad \exists x_1 \, (\\
\qquad\qquad\qquad\qquad\qquad INV_1(main)(MD_1) \wedge \\
\qquad\qquad\qquad\qquad\qquad i_1 < 1000 \wedge \\
\qquad\qquad\qquad\qquad\qquad x = edges[i_1].p \\
\qquad\qquad\qquad\qquad\quad) \wedge \\
\qquad\qquad\qquad\qquad\quad y = edges[i_1].q \\
\qquad\qquad\qquad\qquad) \wedge \\
\qquad\qquad\qquad\qquad MD_2 = upd(MD_1, \&p, MD_1(\&fst(Val_2))) \wedge \\
\qquad\qquad\qquad\qquad (P'(x) \Rightarrow Q'(x, MD_2, Val_2)) \\
\qquad\qquad\qquad\quad) \wedge
\end{array}
$$

$$
\begin{array}{l}
\quad\quad\quad\quad\quad\quad\quad MD_3 = \mathrm{upd}(MD_2, \&q, MD_2(\&\mathrm{fst}(Val_2)) \\
\quad\quad\quad\quad\quad\quad) \wedge \\
\quad\quad\quad\quad\quad\quad (P'(y) \Rightarrow Q'(y, MD_3, Val)) \\
\quad\quad\quad\quad\quad) \wedge \\
\quad\quad\quad\quad\quad (\tau, STD_1) \in \mathrm{pointer}(\mathrm{struct}(\mathrm{trailer})) \wedge \\
\quad\quad\quad\quad\quad (MD_4, V) \in \mathrm{new}(\tau, MD_5) \wedge \\
\quad\quad\quad\quad\quad MD_5 = \mathrm{upd}(MD_4, \&t, MD_4(\&\mathrm{fst}(V))) \\
\quad\quad\quad\quad) \wedge \\
\quad\quad\quad\quad MD_6 = \mathrm{upd}(MD_5, \mathrm{mb}(MD_5(\&t), \mathrm{id}), MD_5(\&q))) \\
\quad\quad\quad) \wedge \\
\quad\quad\quad MD_7 = \mathrm{upd}(MD_6, \mathrm{mb}(MD_6(\&t), \mathrm{next}), MD_6(\mathrm{mb}(MD_6(\&p), \mathrm{trail}))) \\
\quad\quad) \wedge \\
\quad\quad MD_8 = \mathrm{upd}(MD_7, \mathrm{mb}(MD_7(\&p), \mathrm{trail}), MD_7(\&t)) \\
\quad) \wedge \\
\quad MD = \mathrm{upd}(MD_8, \mathrm{mb}(MD_8(\&q), \mathrm{count}), MD_8(\mathrm{mb}(MD_8(\&q), \mathrm{count})) + 1) \wedge \\
\quad i = i_1 + 1 \\
) \Rightarrow \\
INV_1(\mathrm{main})(MD).
\end{array}
$$

Доказательство.

Следующее условие корректности соответствует участку программы после первого цикла до начала второго:

$$
\begin{array}{l}
\exists MD_5 (\\
\quad \exists MD_4 (\\
\quad\quad \exists MD_3 (\\
\quad\quad\quad \exists MD_1\ \exists MD_2\ \exists STD_1\ \exists \tau\ \exists V(\\
\quad\quad\quad\quad INV_1(\mathrm{main})(MD_1) \wedge \\
\quad\quad\quad\quad i \geq 1000 \wedge \\
\quad\quad\quad\quad (\tau, STD_1) \in \mathrm{pointer}(\mathrm{struct}(\mathrm{list})) \wedge \\
\quad\quad\quad\quad (MD_2, V) \in \mathrm{new}(\tau, MD) \\
\quad\quad\quad) \wedge \\
\quad\quad\quad MD_3 = \mathrm{upd}(MD_2, \&res, MD_2(\&\mathrm{fst}(V))) \wedge \\
\quad\quad\quad STD = STD_1 \\
\quad\quad) \wedge \\
\quad\quad MD_4 = \mathrm{upd}(MD_3, \&it, MD_3(\&res)) \\
\quad) \wedge \\
\quad MD_5 = \mathrm{upd}(MD_4, \&p, MD_4(\&head)) \wedge \\
\quad MD = \mathrm{upd}(MD_5, \&head, NULL) \\
) \Rightarrow \\
INV_2(\mathrm{main})(MD).
\end{array}
$$

Очевидно, т. к. MD(&res) = NULL, а из посылки следует, что MD(&head) = NULL.

Применяя правила вывода к операторам второго цикла в силу (П.27), получим ещё два условия корректности:

$$
\begin{aligned}
&\exists MD_4 \,(\\
&\quad \exists MD_3 \,(\\
&\qquad \exists MD_2 \,(\\
&\qquad\quad \exists MD_1 \,(\\
&\qquad\qquad INV_2(main)(MD_1) \wedge\\
&\qquad\qquad MD_1(\&p) \neq MD_1(\&tail) \wedge\\
&\qquad\qquad MD_2 = upd(MD_1, \&q, MD_1(\&p))\\
&\qquad\quad) \wedge\\
&\qquad\quad MD_3 = upd(MD_2, \&p, MD_2(mb(MD_2(\&q), next))) \wedge\\
&\qquad\quad MD_3(mb(MD_3(\&q), count)) = 0\\
&\qquad) \wedge\\
&\qquad MD_4 = upd(MD_3, mb(MD_3(\&q), next), MD_3(\&head))\\
&\quad) \wedge\\
&\quad MD = upd(MD_4, \&head, MD_4(\&q))\\
&) \Rightarrow\\
&INV_2(main)(MD).
\end{aligned}
$$

Доказательство. Из посылки следует, что MD(&head) = MD(&q). Однако MD(mb(MD(&q), count)) = 0, поэтому MD(mb(MD(&head), count)) = 0. Следовательно, для первого элемента кортежа ttuple(MD(&head)) заключение условия корректности истинно.

В силу того, что mb(MD(&head), next) = mb(MD(&q), next) = MD_1(&head), получаем ttuple(MD(&head)) = con(tuple(MD(&head), MD(&head)), ttuple(MD_1(head))). А т. к. $\forall$f (f $\in$ ttuple(MD_1(&head)) $\Rightarrow$ MD_1(mb(f, count)) = 0), то заключение условия корректности истинно и для остальных элементов кортежа ttuple.

MD(&res) = MD_1(&res), и доказательство завершено.

$$
\begin{aligned}
&\exists MD_2 \,(\\
&\quad \exists MD_1 \,(\\
&\qquad INV_2(main)(MD_1) \wedge\\
&\qquad MD_1(\&p) \neq MD_1(\&tail)\\
&\quad) \wedge\\
&\quad MD_2 = upd(MD_1, \&q, MD_1(\&p)) \wedge
\end{aligned}
$$

$MD = upd(MD_2, \&p, MD_2(mb(MD_2(\&q), next)) \wedge$
$MD(mb(MD(\&q), count)) \neq 0$
$) \Rightarrow$
$INV_2(main)(MD)$.

В этом случае $MD(\&head) = MD_1(\&head)$, и $MD(\&res) = MD_1(\&res)$, поэтому истинность очевидна.

Следующее условие корректности получаем на участке программы между вторым и третьим циклами.

$\exists MD_1$ (
$INV_2(main)(MD_1) \wedge$
$MD_1(\&p) = MD_1(\&tail) \wedge$
$MD = upd(MD_1, \&q, MD_1(\&head))$
$) \Rightarrow$
$INV_3(main)(MD)$.

Пусть $MD_1(\&head) \neq NULL$, тогда истинно $\forall f$ ($f \in ttuple(MD_1(\&head)) \Rightarrow MD_1(mb(f, count)) = 0$). Но $MD(\&q) = MD_1(\&head)$, а значит, $MD(\&q) \neq NULL$. Следовательно, $MD(mb(MD(\&q), count)) = 0$.

Пусть $MD_1(\&head) = NULL$. Т. к. $MD(\&q) = MD_1(\&head)$, то $MD(\&q) = NULL$, следовательно, посылка в импликации заключения условия корректности ложна, а значит, импликация истинна.

$MD(\&res) = MD_1(\&res)$ и доказательство завершено.

Далее мы переходим, к рассмотрению третьего цикла, в теле которого имеется ещё один вложенный. В соответствии с правилом (П.28) получим условие корректности:

$\exists MD_6$ (
$\exists MD_5$ (
$\exists MD_4$ (
$\exists MD_2\ \exists MD_3\ \exists STD_1\ \exists \tau\ \exists V$ (
$\exists MD_1$ (
$INV_3(main)(MD_1) \wedge$
$MD_1(\&q) \neq NULL \wedge$
$MD_2 = upd(MD_1, mb(MD_1(\&it), key), MD_1(mb(MD_1(\&q), key)))$
$) \wedge$
$(\tau, STD_1) \in pointer(struct(list)) \wedge$
$(MD_3, V) \in new(\tau, MD) \wedge$
$MD_4 = upd(MD_3, mb(MD_3(\&it), next), MD_3(\&fst(V))) \wedge$
$STD = STD_1$

```
    ) ∧
    MD5 = upd(MD4, &it, MD4(mb(MD4(&it), next)))
  ) ∧
  MD6 = upd(MD5, &t, MD5(mb(MD5(&q), trail)))
) ∧
MD = upd(MD6, &q, MD6(mb(MD6(&q), next)))
) ⇒
INV4(main)(MD).
```

Доказательство. На этом участке список ведущих, на начало которого указывает head, не изменяется, поэтому свойство MD(mb(f, count)) = 0 сохраняется для любого его элемента f.

Так как в выходной список, на который указывает res, добавлен элемент MD_1(mb(MD_1(&q), key)), то predcount(MD(mb(MD(mb(MD(&t), id)), key)), MD(&res)) уменьшилась на 1, что и утверждается в последнем конъюнкте заключения.

Докажем истинность предиката topsort(MD(&res)). В выходной список добавилась вершина с номером MD_1(&q), т. е. имеем ttuple(MD(&res)) = con(ttuple(MD_1(&res)), ttuple(MD_1(&it))). По определению истинность предиката topsort означает, что для любой вершины выходного списка из неё нет дуг ни в одну предшествующую. Предположим противное: из вершины с номером MD_1(&q) есть дуга в некоторую вершину из списка ttuple(MD_1(&res)). Но тогда это противоречит тому, что у всех вершин выходного списка в списке ведущих поле count = 0.

Далее при рассмотрении внутреннего тела цикла мы получим ещё два условия корректности в силу правила (П.27).

```
∃MD5 (
  ∃MD4 (
    ∃MD3 (
      ∃MD2 (
        ∃MD1 (
          INV4(main)(MD1) ∧
          MD1(&t) ≠ NULL ∧
          MD2 = upd(MD1, &p, MD1(mb(MD1(&t), id)))
        ) ∧
        MD3 = upd(MD2, mb(MD2(&p), count), MD2(mb(MD2(&p), count)) - 1)
      ) ∧
      MD3(mb(MD3(&p), count)) = 0 ∧
      MD4 = upd(MD3, mb(MD3(&p), next), MD3(&q))
    ) ∧
```

$$
\begin{array}{l}
\quad MD_5 = upd(MD_4, \&q, MD_4(\&p)) \\
\quad) \wedge \\
\quad MD = upd(MD_5, \&t, MD_5(mb(MD_5(\&t), next))) \\
) \Rightarrow \\
INV_4(main)(MD).
\end{array}
$$

Доказательство. В данном случае в список ведущих был добавлен новый элемент, поэтому нужно проверить, что $MD(mb(MD(\&q), count)) = 0$. Действительно, $MD(\&q) = MD(\&p) = MD_3(\&p)$, но $MD_3(mb(MD_3(\&p), count)) = 0$.

Выходной список, на который указывает res, не изменился, поэтому свойство $topsort(MD(\&res))$ выполнено.

По этой же причине выполнено равенство в последнем конъюнкте.

$$
\begin{array}{l}
\exists MD_3 (\\
\quad \exists MD_2 (\\
\quad\quad \exists MD_1 (\\
\quad\quad\quad INV_4(main)(MD_1) \wedge \\
\quad\quad\quad MD_1(\&t) \neq NULL \wedge \\
\quad\quad\quad MD_2 = upd(MD_1, \&p, MD_1(mb(MD_1(\&t), id))) \\
\quad\quad) \wedge \\
\quad\quad MD_3 = upd(MD_2, mb(MD_2(\&p), count), MD_2(mb(MD_2(\&p), count)) - 1) \\
\quad) \wedge \\
\quad MD_3(mb(MD_3(\&p), count)) \neq 0 \wedge \\
\quad MD = upd(MD_3, \&t, MD_3(mb(MD_3(\&t), next))) \\
) \Rightarrow \\
INV_4(main)(MD).
\end{array}
$$

Доказательство. В этом случае $MD(\&q) = MD_1(\&q)$, поэтому список ведущих, на который указывает head, не изменился. Следовательно, свойство $MD(mb(f, count)) = 0$ выполнено.

Аналогично выходной список, на который указывает res, не изменился, поэтому свойство $topsort(MD(\&res))$ сохраняется.

Равенство в последнем конъюнкте заключения очевидно.

Очередное условие корректности соответствует выходу из внутреннего цикла. Поскольку на оставшемся участке до конца внешнего цикла нет операторов, то по правилу (П.24) сразу же получаем:

$$INV_4(main)(MD) \wedge MD(\&t) = NULL \Rightarrow INV_3(main)(MD).$$

Истинность очевидна.

Наконец, выводим последнее условие корректности, соответствующее выходу из последнего цикла.

$INV_3(main)(MD) \wedge MD(\&q) = NULL \Rightarrow SP_{post}(main)(MD)$.

Истинность также очевидна.

5. ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена программа топологической сортировки на языке C. С помощью разработанной смешанной аксиоматической семантики была успешно доказана её частичная корректность.

СПИСОК ЛИТЕРАТУРЫ

1. Непомнящий В. А., Ануреев И. С., Михайлов И. Н., Промский А. В. Ориентированный на верификацию язык C-light // Системная информатика: Сб. науч. тр. – Новосибирск: Издательство СО РАН. – 2004. – Вып. 9: Формальные методы и модели информатики. – С. 51–134.
2. Марьясов И. В. На пути к автоматической верификации C-light программ. Смешанная аксиоматическая семантика языка C-kernel. – Новосибирск, 2008. – 32 с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 150).
3. Вирт Н. Алгоритмы и структуры данных. – Досса: Хамарайан, 1997.
4. Марьясов И. В. Автоматическая верификация программ на языке C-light // VIII Всерос. конф. молодых учёных по математическому моделированию и информационным технологиям: Программа и тезисы докладов. – Новосибирск: Институт вычислительных технологий СО РАН, 2007. – С. 103.
5. Марьясов И. В. Автоматическая верификация программ на языке C-light // Технологии Microsoft в теории и практике программирования: конференция-конкурс работ студентов, аспирантов и молодых учёных: тезисы докладов. – Новосибирск: Новосибирский государственный университет, 2007. – С. 25–27.
6. Непомнящий В. А., Рякин, О. М. Прикладные методы верификации программ. – М.: Радио и связь, 1988.
7. Hoare C. A. R. An axiomatic basis for computer programming // Communs. of the ACM. – 1969. – Vol. 12, N 1. – P. 576–580.
8. Maryasov I. V. Towards automatic verification of C-light programs. Mixed axiomatic semantics of C-kernel language // Perspectives of Systems Informatics (PSI): A. Ershov 7th Int. Conf.: Int. workshop on Program Understanding. – Novosibirsk, 2009. – P. 44–52.

Приложение 1

Список правил смешанной аксиоматической семантики языка C-kernel

Вызов функции. Правило вывода для функции, обозначаемой выражением e_0, в случае, когда она возвращает значение, имеет вид:

$$\frac{\begin{array}{c}E, SP \vdash \{\exists Val'\ \alpha \wedge P'(x_1 \leftarrow cast(val(e_1, MD, STD), type(e_1), \tau_1), \ldots, \\ x_n \leftarrow cast(val(e_n, MD, STD), type(e_n, MD, STD), \tau_n) \Rightarrow \\ Q'(x_1 \leftarrow cast(val(e_1, MD, STD), type(e_1), \tau_1), \ldots, \\ x_n \leftarrow cast(val(e_n, MD, STD), type(e_n), \tau_n)\}\ A;\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{e = e_0(e_1, \ldots, e_n);}\ A;\ \{Q\}} \qquad (П.1)$$

SP_{fun} – спецификация функции e_0;
$P' = fst(SP_{fun}(val(e_0, MD, STD)))(MD, STD, Val)$ – предусловие функции e_0;
$Q' = snd(SP_{fun}(val(e_0, MD, STD)))(MD, STD, Val)$ – постусловие функции e_0;
$x_1, \ldots, x_n$ – формальные параметры функции e_0;
$\alpha = \exists MD'\ P(MD \leftarrow MD')(Val \leftarrow Val') \wedge MD = upd(MD', loc(val(e, MD', STD)), cast(fst(Val), snd(Val), type(e)))$, если e – переменная обычного вида;
$\alpha = \exists e'\ P(e \leftarrow e')(Val \leftarrow Val') \wedge e = cast(fst(Val), snd(Val), type(e))$, если e – простая переменная паскалевского вида;
$\alpha = \exists v'\ P(v \leftarrow v')(Val \leftarrow Val') \wedge v = upd(v', val(i, MD, STD), cast(fst(Val), snd(Val), type(v'))$, если $e = v[i]$ – массив паскалевского вида;
$\alpha = \exists s'\ P(s \leftarrow s')(Val \leftarrow Val') \wedge s = upd(s', t, cast(fst(Val), snd(Val), type(s')))$, если $e=s.t$ – структура паскалевского вида.

MD', e', v', s' – новые переменные соответствующих типов.

Правило вывода для функции, обозначаемой выражением e_0, в случае, когда она не возвращает значение, имеет вид:

$$\frac{\begin{array}{c}E, SP \vdash \{P \wedge P'(x_1 \leftarrow cast(val(e_1, MD, STD), type(e_1), \tau_1), \ldots, \\ x_n \leftarrow cast(val(e_n, MD, STD), type(e_n), \tau_n) \Rightarrow \\ Q'(x_1 \leftarrow cast(val(e_1, MD, STD), type(e_1), \tau_1), \ldots, \\ x_n \leftarrow cast(val(e_n, MD, STD), type(e_n), \tau_n)\}\ A;\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{e_0(e_1, \ldots, e_n);}\ A;\ \{Q\}} \qquad (П.2)$$

Операция new.

$$\frac{\begin{array}{c}E, SP \vdash \{\exists MD'\ \exists STD'\ \exists MD''\ \exists STD''\ \exists \tau\ \exists V \\ P(MD \leftarrow MD')(STD \leftarrow STD') \wedge (\tau, STD'') \in logtype(e', MD, STD) \wedge \\ (MD'', V) \in new(\tau, MD) \wedge \\ MD = upd(MD'', addr(val(e, MD'', STD'')), cast(fst(V), snd(V), type(e))) \\ \wedge\ STD = STD''\}\ A;\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{e = new\ e';}\ A;\ \{Q\}} \qquad (П.3)$$

Операция delete.

$$\frac{E, SP \vdash \{\exists MD'\ \exists MD''\ P(MD \leftarrow MD') \wedge MD'' = delete(MD', type(e), val(e, MD', STD) \wedge MD = MD''\}\ A\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{delete\ e;}\ A;\ \{Q\}} \qquad (П.4)$$

Операция присваивания. Пусть выражение e_0 не содержит вызовов функций, операторов new и операторов приведения.

$$\frac{E, SP \vdash \{\exists MD'\ P(MD \leftarrow MD') \wedge (MD = upd(MD', addr(e, MD'), cast(val(e_0, MD', STD), type(e_0), type(e))))\}\ A;\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{e = e_0;}\ A;\ \{Q\}} \qquad (П.5)$$

где e не является переменной паскалевского вида.

В случае, когда e – простая переменная паскалевского вида, правило имеет вид:

$$\frac{E, SP \vdash \{\exists e'\ P(e \leftarrow e') \wedge (e = cast(val(e_0(e \leftarrow e'), MD, STD), type(e_0(e \leftarrow e')), type(e)))\}\ A;\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{e = e_0;}\ A;\ \{Q\}} \qquad (П.6)$$

Если e = v[i] – элемент массива паскалевского вида, то

$$\frac{\text{E, SP} \vdash \{\exists v'\ P(v \leftarrow v') \wedge (v = upd(v', val(i, MD, STD), cast(val(e_0(v \leftarrow v'), MD, STD), type(e_0(v \leftarrow v')), type(v[i])))\}\ A;\ \{Q\}}{\text{E, SP} \vdash \{P\}\ \mathbf{v[i] = e_0;}\ A;\ \{Q\}} \quad (П.7)$$

Если e = s.t – структура паскалевского вида, то

$$\frac{\text{E, SP} \vdash \{\exists s'\ P(s \leftarrow s') \wedge (s = upd(s', t, cast(val(e_0(s \leftarrow s'), MD, STD), type(e_0(s \leftarrow s')), type(s.t)))\}\ A;\ \{Q\}}{\text{E, SP} \vdash \{P\}\ \mathbf{s.t = e_0;}\ A;\ \{Q\}} \quad (П.8)$$

Декларации переменных. Правило вывода для декларации переменной, не являющейся переменной паскалевского вида, без инициализатора имеет вид:

$$\frac{\begin{array}{c}\text{E, SP} \vdash \{\exists MD'\ \exists STD'\ \exists nc\ \exists \tau\ \exists V\ \exists MD''\ \exists STD'' \\ P(MD \leftarrow MD')(STD \leftarrow STD') \wedge \\ (\tau, STD'') \in logtype(tp(e), MD', STD') \wedge MD'(nc) = \perp \wedge \\ (MD'', V) \in init(\tau, storage(e), upd(MD', nc, \omega)) \wedge \\ MD = upd(MD'', nc, fst(V)) \wedge STD = STD''\}\ A\ \{Q\}\end{array}}{\text{E, SP} \vdash \{P\}\ \mathbf{e;}\ A;\ \{Q\}} \quad (П.9)$$

Для простой переменной e паскалевского вида правило имеет вид:

$$\frac{\text{E, SP} \vdash \{\exists e'\ P(e \leftarrow e') \wedge e = defaultValue(\tau, storage)(e \leftarrow e')\}\ A\ \{Q\}}{\text{E, SP} \vdash \{P\}\ \mathbf{storage\ \tau\ e;}\ A;\ \{Q\}} \quad (П.10)$$

Если в декларации объявляется массив v паскалевского вида, то

$$\frac{\text{E, SP} \vdash \{\exists v'\ P(v \leftarrow v') \wedge v=((dV, \ldots, dV), \ldots(dV, \ldots, dV))\}\ A\ \{Q\}}{\text{E, SP} \vdash \{P\}\ \mathbf{storage\ \tau[n_1, \ldots, n_k]\ v;}\ A;\ \{Q\}} \quad (П.11)$$

где dV = defaultValue(τ, storage)(v ← v').

В случае, когда объявляется структура s паскалевского вида, правило имеет вид:

$$\frac{E, SP \vdash \{\exists s'\ P(s \leftarrow s') \wedge s = (defaultValue(\tau_1, storage)(s \leftarrow s'), \ldots, defaultValue(\tau_n, storage)(s \leftarrow s'))\}\ A\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{storage\ struct\ s\ \{\tau_1\ t_1; \ldots; \tau_n\ t_n;\};}\ A;\ \{Q\}} \quad (П.12)$$

Правило вывода для декларации переменной, не являющейся переменной паскалевского вида, с инициализатором имеет вид:

$$\frac{\begin{array}{c}E, SP \vdash \{\exists MD'\ \exists STD'\ \exists nc\ \exists \tau\ \exists V\ \exists MD''\ \exists STD'' \\ P(MD \leftarrow MD')(STD \leftarrow STD') \wedge \\ (\tau, STD'') \in logtype(tp(e), MD', STD') \wedge MD'(nc) = \perp \wedge \\ (MD'', V) \in init(\tau, e_0, upd(MD', nc, \omega)) \wedge \\ MD = upd(MD'', nc, fst(V)) \wedge STD = STD''\}\ A\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{e = e_0;}\ A;\ \{Q\}} \quad (П.13)$$

Для простой переменной e паскалевского вида правило имеет вид:

$$\frac{E, SP \vdash \{\exists e'\ P(e \leftarrow e') \wedge e = e_0(e \leftarrow e')\}\ A\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{storage\ \tau\ e = e_0;}\ A;\ \{Q\}} \quad (П.14)$$

Если в декларации объявляется массив v паскалевского вида, то

$$\frac{E, SP \vdash \{\exists v'\ P(v \leftarrow v') \wedge v = ((v_{0\ldots0}(v \leftarrow v'), \ldots, v_{0\ldots0\ n1-1}(v \leftarrow v')), \ldots, (v_{nk-1\ldots nk-1\ 0}(v \leftarrow v'), \ldots, v_{nk-1\ldots nk-1}(v \leftarrow v')))\}\ A\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{storage\ \tau[n_1, \ldots, n_k]\ v = \{\{v_{0\ldots0}, \ldots, v_{0\ldots0\ n1-1}\}, \ldots, \{v_{nk-1\ldots nk-1\ 0}, \ldots, v_{nk-1\ldots nk-1}\}\};}\ A;\ \{Q\}} \quad (П.15)$$

В случае, когда объявляется структура s паскалевского вида, правило имеет вид:

$$\frac{E, SP \vdash \{\exists s'\ P(s \leftarrow s') \wedge s = (v_1(s \leftarrow s'), \ldots, v_n(s \leftarrow s'))\}\ A\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{storage\ struct\ s\ \{}\tau_1\ t_1 = v_1\mathbf{;\ \ldots;}\ \tau_n\ t_n = v_n\mathbf{;\};}\ A;\ \{Q\}} \quad (П.16)$$

Декларации типов.

$$\frac{\begin{array}{c}E, SP \vdash \{\exists STD'\ \exists\tau\ \exists STD''\ P(STD \leftarrow STD') \wedge \\ (\tau, STD'') \in logtype(e, MD, STD') \\ \wedge\ STD = upd(STD'', id(e), \tau)\}\ A;\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{typedef\ e;}\ A;\ \{Q\}} \quad (П.17)$$

Декларации функций.

$$\frac{\begin{array}{c}E, SP \vdash \{\exists MD'\ \exists STD'\ \exists\tau'\ \exists STD''\ \exists nc \\ P(MD \leftarrow MD')(STD \leftarrow STD') \wedge \\ (\tau', STD'') \in logtype(\tau\ f(\tau_1\ x_1, \ldots, \tau_n\ x_n), MD', STD') \wedge \\ MD'(nc) = \perp \wedge MD = upd(MD', nc, (f, [x_1, \ldots, x_n], S)) \wedge \\ STD = STD''\}\ A;\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ \boldsymbol{\tau\ f(\tau_1\ x_1, \ldots, \tau_n\ x_n)\{S\}}\ A;\ \{Q\}} \quad (П.18)$$

Помеченный оператор.

$$\frac{\begin{array}{c}(f, \tau, B, cur, L), SP \vdash \{P\}\ A;\ \{Q\} \\ (f, \tau, B, cur, \omega), SP \vdash \{SP_{lab}(L_1)\}\ S;\ A;\ \{Q\}\end{array}}{(f, \tau, B, cur, L), SP \vdash \{P\}\{SP_{lab}(L_1)\}\ L_1\colon S;\ A;\ \{Q\}}\ L \neq L_1 \quad (П.19)$$

$$\frac{\begin{array}{c}SP_{fun} \models P \Rightarrow SP_{lab}(L) \\ (f, \tau, B, cur, \omega), SP \vdash \{SP_{lab}(L)\}\ S;\ A;\ \{Q\}\end{array}}{(f, \tau, B, cur, E_5), SP \vdash \{P\}\{SP_{lab}(L)\}\ L\colon S;\ A;\ \{Q\}}\ \begin{array}{c}E_5 = \omega \\ \text{или} \\ E_5 = L\end{array} \quad (П.20)$$

Блок.

$$\frac{(f, \tau, B, id, E_5), SP \vdash \{P\}\ S;\ A;\ \{Q\}}{(f, \tau, B, cur, E_5), SP \vdash \{P\}\{S\}_{id};\ A;\ \{Q\}} \quad (П.21)$$

$$\frac{(f, \tau, B, cur, E_5), SP \vdash \{P\}\ A;\ \{Q\}}{(f, \tau, B, id, E_5), SP \vdash \{P\}\ blockEnd(cur);\ A;\ \{Q\}}\ id \neq E_5 \quad (П.22)$$

$$\frac{SP_{fun} \models P \Rightarrow SP_{post}(f)}{(f, \tau, B, id(f), id(f)), SP \vdash \{P\}\ blockEnd(cur);\ A;\ \{Q\}} \quad (П.23)$$

Пустой оператор.

$$\frac{E, SP \vdash \{P\}\ A;\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{;}\ A;\ \{Q\}} \quad (П.24)$$

$$\frac{SP_{fun} \models P \Rightarrow Q}{(f, \tau, B, cur, \omega), SP \vdash \{P\}\ \mathbf{;}\ \{Q\}} \quad (П.25)$$

$$\frac{SP_{fun} \models P \Rightarrow SP_{lab}(L)}{(f, \tau, B, cur, L), SP \vdash \{P\}\ \mathbf{;}\ \{Q\}} \quad (П.26)$$

Условный оператор.

$$\frac{\begin{array}{l}E, SP \vdash \{P \wedge cast(val(e, MD, STD), type(e), int) \neq 0\}\ S_1;\ A;\ \{Q\} \\ E, SP \vdash \{P \wedge cast(val(e, MD, STD), type(e), int) = 0\}\ S_2;\ A;\{Q\}\end{array}}{E, SP \vdash \{P\}\ \mathbf{if\ (e)\ S_1\ else\ S_2;}\ A;\ \{Q\}} \quad (П.27)$$

Цикл.

$$\frac{\begin{array}{c}SP_{fun} \models P \Rightarrow INV \\ E, SP \vdash \{INV \wedge cast(val(e, MD, STD), type(e), int) \neq 0\}\ S;\ \{INV\} \\ E, SP \vdash \{INV \wedge cast(val(e, MD, STD), type(e), int) = 0\}\ A\ \{Q\}\end{array}}{E, SP \vdash \{P\}\{INV\}\ \textbf{while (e) S;}\ A;\ \{Q\}} \quad (П.28)$$

Операторы перехода.

$$\frac{(f, \tau, B, cur, L), SP \vdash \{P\}\ A;\ \{Q\}}{(f, \tau, B, cur, \omega), SP \vdash \{P\}\ goto\ L;\ A;\ \{Q\}} \quad (П.29)$$

$$\frac{(f, \tau, B, cur, L), SP \vdash \{P\}\ A;\ \{Q\}}{(f, \tau, B, cur, L), SP \vdash \{P\}\ T;\ A;\ \{Q\}} \quad (П.30)$$

где T не является помеченным оператором и не является blockEnd(cur).

$$\frac{(f, \tau, B, cur, id(f)), SP \vdash \{P\}\ A\ \{Q\}}{(f, \tau, B, cur, \omega), SP \vdash \{P\}\ return;\ A\ \{Q\}} \quad (П.31)$$

$$\frac{(f, \tau, B, cur, id(f)), SP \vdash \{P\}\ A\ \{Q\}}{(f, \tau, B, cur, id(f)), SP \vdash \{P\}\ T;\ A\ \{Q\}} \quad (П.32)$$

$$\frac{\begin{array}{c}(f, \tau, B, cur, id(f)), SP \vdash \{\exists Val'\ P(Val \leftarrow Val') \wedge \\ Val = (cast(val(e, MD, STD), type(e), \tau), \tau)\}\ A\ \{Q\}\end{array}}{(f, \tau, B, cur, \omega), SP \vdash \{P\}\ return\ e;\ A\ \{Q\}} \quad (П.33)$$

где T не является помеченным оператором и не является blockEnd(…).

Правило консеквенции.

$$\frac{SP_{fun} \models P \Rightarrow R \qquad E, SP \vdash \{R\}\ S\ \{T\} \qquad SP_{fun} \models T \Rightarrow Q}{E, SP \vdash \{P\}\ S\ \{Q\}} \qquad (П.34)$$

Последовательность операторов. Пусть T и T' – непустые последовательности операторов и вспомогательных конструкций.

$$\frac{E, SP \vdash \{P\}\ T\ \{R\} \qquad E, SP \vdash \{R\}\ T'\ \{Q\}}{E, SP \vdash \{P\}\ T\ T'\ \{Q\}} \qquad (П.35)$$

Программа. Правила для программы Prgm(T), состоящей из последовательности деклараций T, имеет вид:

$$\frac{\begin{array}{c}(f, \tau_f, S_f, bid_f, \omega), ((SP_{pre}, SP_{post}), SP^f_{lab}) \vdash \{SP_{pre}(f)(x_1, \ldots, x_n)(MD, STD, Val)\}\ S_f;\ blockEnd(bid_f)\ \{SP_{post}(f)(x_1, \ldots, x_n)(MD, STD, Val)\} \\ \text{по всем именам функций f, определённых в T, кроме main,} \\ (main, \tau_{main}, S_{main}, bid_{main}, \omega), SP \vdash \{P\}\ T\ S_{main};\ blockEnd(bid_{main})\ \{Q\}\end{array}}{E, SP \vdash \{P\}\ Prgm(T)\ \{Q\}} \qquad (П.36)$$

где $x_1, \ldots, x_n$ – формальные параметры функции f.

Приложение 2

Текст программы топологической сортировки

Исходная программа имеет вид:

```
typedef struct {int p,q;} edge;
typedef struct leader* LPtr;
typedef struct trailer* TPtr;
typedef struct list* ListPtr;
struct leader
{
  int key, count;
  TPtr trail;
  LPtr next;
};
struct trailer
{
  LPtr id;
  TPtr next;
};
struct list
{
  int key;
  struct list* next;
};
LPtr p, q, head, tail;
TPtr t;
ListPtr res, it;
int i, x, y;
edge edges[num];
/ % SPpre(find) % /
LPtr find(int w)
{
  LPtr h;
  h=head;
  tail->key=w;
  / % INV(find) % /
      while (h->key!=w) h=h->next;
  if (h==tail)
  {
```

```
        tail=new leader;
        h->count=0;
        h->trail=NULL;
        h->next=tail;
  }
  return h;
}
/ % SP_post(find) % /

/ % SP_pre(main) % /
ListPtr main()
{

  head=new leader;
  tail=head;
  / % INV_1(main) % /
  for(i=0;i<1000;i++)
  {
        x=edges[i].p;
        y=edges[i].q;
        p=find(x);
        q=find(y);
        t=new trailer;
        t->id=q;
        t->next=p->trail;
        p->trail=t;
        q->count++;
  }
  res=new struct list;
  it=res;
  p=head;
  head=NULL;
  / % INV_2(main) % /
  while (p!=tail)
  {
        q=p;
        p=q->next;
        if (q->count==0)
        {
              q->next=head;
              head=q;
        }
  }
```

```
  q=head;
  /% INV3(main) %/
  while (q!=NULL)
  {
        it->key=q->key;
        it->next=new struct list;
        it=it->next;
        t=q->trail;
        q=q->next;
        /% INV4(main) %/
        while (t!=NULL)
        {
              p=t->id;
              p->count--;
              if (p->count==0)
              {
                    p->next=q;
                    q=p;
              }
              t=t->next;
        }
  }
  return res;
}
/% SPpost(main) %/
```

Соответствующая ей программа на C-kernel имеет вид:

```
typedef struct {int p; int q;} edge;
typedef struct leader* LPtr;
typedef struct trailer* TPtr;
typedef struct list* ListPtr;
struct leader
{
  int key; int count;
  TPtr trail;
  LPtr next;
};
struct trailer
{
  LPtr id;
```

```
  TPtr next;
};
struct list
{
  int key;
  struct list* next;
};
LPtr p; LPtr q; LPtr head; LPtr tail;
TPtr t;
ListPtr res; ListPtr it;
int i; int x; int y;
edge edges[1000];
/ % SP_pre(find) % /
LPtr find(int w)
{
  LPtr h;
  h=head;
  tail->key=w;
  / % INV(find) % /
      while (h->key!=w) {h=h->next};
  if (h==tail)
  {
        tail=new leader;
        h->count=0;
        h->trail=NULL;
        h->next=tail;
  }
  else {}
  return h;
}
/ % SP_post(find) % /

/ % SP_pre(main) % /
ListPtr main()
{

  head=new leader;
  tail=head;
  i=0;
      / % INV_1(main) % /
  while(i<1000)
  {
```

```
        x=edges[i].p;
        y=edges[i].q;
        p=find(x);
        q=find(y);
        t=new trailer;
        t->id=q;
        t->next=p->trail;
        p->trail=t;
        q->count=q->count+1;
        i=i+1;
}
res=new struct list;
it=res;
p=head;
head=NULL;
    /* INV2(main) */
while (p!=tail)
{
        q=p;
        p=q->next;
        if (q->count==0)
        {
                q->next=head;
                head=q;
        }
        else {}
}
q=head;
    /* INV3(main) */
while (q!=NULL)
{
        it->key=q->key;
        it->next=new struct list;
        it=it->next;
        t=q->trail;
        q=q->next;
                /* INV4(main) */
        while (t!=NULL)
        {
                p=t->id;
                p->count=p->count-1;
                if (p->count==0)
```

```
                {
                        p->next=q;
                        q=p;
                }
                else {}
                t=t->next;
        }
  }
  return res;
}
```

/ % SP_{post}(main) % /

И. В. Марьясов

ПРИМЕНЕНИЕ СМЕШАННОЙ АКСИОМАТИЧЕСКОЙ СЕМАНТИКИ ЯЗЫКА C-KERNEL К ВЕРИФИКАЦИИ ПРОГРАММЫ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ

Препринт
155

Рукопись поступила в редакцию 25.04.10
Редактор Т. М. Бульонкова
Рецензент И.С. Ануреев

Подписано в печать 15.06.10
Формат бумаги 60 × 84 1/16 Объем 2.0 уч.-изд.л., 2.2 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42