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

А. Е. Рязанов

СИСТЕМА БУЦЕФАЛ: КОМБИНИРОВАНИЕ ДЕДУКТИВНЫХ ПРОЦЕДУР И ОПИСАНИЕ СТРАТЕГИЙ ПОИСКА ДОКАЗАТЕЛЬСТВ

Препринт
50

Новосибирск 1998

В работе представлен прототип системы поддержки доказательств, основанный на понятии стратегии. Основной акцент делается на описании языка задания стратегии, положенного в основу системы. Описывается декларативная и неформальная операционная семантика языка. Значительная доля примеров, иллюстрирующих возможности языка, посвящена комбинированию разнородных дедуктивных процедур в рамках единой стратегии.

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

A. E. Ryazanov

SYSTEM BUTSEPHALAS: DEFINING PROOF-SEARCH STRATEGIES AND COMBINING DEDUCTIVE PROCEDURES

Preprint
50

Novosibirsk 1998

We present a semiautomatic theorem prover based on the notion of strategy. Main emphasis is made on description of the strategy definition language which is the key point in design of the system. Declarative and informal operational semantics of the language are described. Substantial number of examples illustrating features of the language is devoted to combining heterogeneous deductive procedures into a single proof-search strategy.

ВВЕДЕНИЕ

БУЦЕФАЛ — прототип системы, ориентированной на автоматическую поддержку процесса поиска доказательств для решения задач верификации программного и аппаратного обеспечения. Формальная верификация выдвигает специфические требования к таким системам. Перечислим некоторые из них, которые, на наш взгляд, являются наиболее существенными и обязательно должны находить отражение в дизайне систем поддержки доказательств.

Дедуктивные задачи, возникающие в формальной верификации, могут быть охарактеризованы следующим образом: преобладающие простые задачи, обычно легко решаемые автоматически, сочетаются с задачами средней сложности, которые плохо или совсем не решаются автоматически (типичный пример — использование индукции), но достаточно легко решаются, если пользователь имеет возможность направлять поиск вывода. Исходя из этого полагаем, что в системе поддержки доказательств, ориентированной на верификацию, процесс построения доказательств должен быть в значительной степени автоматизирован, но должны присутствовать также достаточно мощные средства управления поиском доказательств, чтобы в предельном случае процесс поиска вывода мог вырождаться в интерактивное построение доказательства. При этом мощь аппарата управления поиском хотелось бы сочетать с наглядностью.

Проблемная ориентированность верификации подразумевает использование разнородных дедуктивных средств — логик и дедуктивных процедур (разрешающих, упрощающих и т.п.). Поэтому хотелось бы, чтобы наша гипотетическая система обладала достаточной гибкостью для реализации разных логик и дедуктивных процедур и предоставляла простые и наглядные средства их комбинирования.

При решении дедуктивных задач, связанных с верификацией, от системы требуется не просто положительный или отрицательный ответ на вопрос о корректности какого-то утверждения, а построение доказательства (или хотя бы наброска, по которому можно восстановить доказательство), которое может быть подвергнуто простой механической проверке и использовано в качестве документации. БУЦЕФАЛ — это результат попытки разработать систему, удовлетворяющую перечисленным выше требованиям. Главный акцент в дизайне системы сделан на управление поиском вывода с помощью описания стратегий и комбинирование дедуктивных процедур. В основе системы лежит простой и

наглядный язык описания стратегий, который будет формально описан в следующем параграфе. Кроме того, БУЦЕФАЛ яляется логиконезависимой системой. Строго говоря, БУЦЕФАЛ — не доказыватель, а среда для быстрого построения доказывателей (в англоязычной литературе для таких систем принят термин logical framework). Та или иная логика, поиск вывода в которой мы хотели бы автоматизировать, реализуется фиксированием синтаксиса и написанием базовых дедуктивных процедур на языке Concurrent ML, являющемся параллельным расширением языка Standard ML (см. [7], необходимую информацию о Standard ML можно найти в [6]). Такой подход в современных системах поддержки доказательств используется весьма широко (LCF [2], Isabelle [3, 4], PVS [5]), что, в частности, позволяет программировать достаточно эффективные примитивные процедуры с тем, чтобы в последствии, комбинируя эти процедуры, решать сложные задачи. С точки зрения пользователя, БУЦЕФАЛ может рассматриваться и как оболочка для быстрого построения доказывателей, и как библиотека на языке CML, которую можно использовать для программирования доказывателей. Наконец заметим, что семантика языка описания стратегий, положенного в основу системы, подразумевает определенный параллелизм. С одной стороны, этим обусловлен выбор CML в качестве языка реализации. С другой стороны, это позволяет предположить возможность реализации системы с такой идеологией на распределенных и многопроцессорных архитектурах.

1. ФОРМАЛЬНЫЕ ОСНОВЫ: ЯЗЫК ОПИСАНИЯ СТРАТЕГИЙ

1.1. Обозначения и соглашения

Пусть $\mathcal{N}$ — множество натуральных чисел, S — произвольное множество. Мультимножеством над S будем называть любое множество пар вида $< x, n >$, где $x \in S, n \in \mathcal{N}$, не содержащее пар с совпадающими n. Для упрощения нотации при указании элементов мультимножества числа n будут опускаться. Множество всех конечных мультимножеств над S и множество всех конечных непустых мультимножеств над S будем обозначать $Mult(S)$ и $Mult^{+}(S)$ соответственно.

Множество всех конечных последовательностей с элементами из S обозначим $Seq(S)$. Если $s \in Seq(S), x \in S$, то $x.s$ обозначает последовательность, полученную из s добавлением x в качестве первого элемента.

Последовательность нулевой длины обозначим λ.

Будем считать, что на конечных непустых множествах (и мультимножествах в том числе) определена вычислимая операция ch выбора некоторого элемента.

1.2. Синтаксис и статическая семантика

В системе БУЦЕФАЛ ключевым понятием является стратегия. Пользователь задает логику, вывод в которой нужно автоматизировать, фиксируя ее синтаксис и программируя атомарные стратегии, а затем, для решения конкретных задач, описывает стратегии поиска вывода, комбинируя атомарные стратегии с помощью операций языка описания стратегий. Опишем синтаксис ядра этого языка. Элементами синтаксической категории < ид. > будут произвольные алфавитно-цифровые идентификаторы. В дальнейшем множество всех идентификаторов будем обозначать V. Определим синтаксическую категорию ориентированных переменных:

$$\begin{aligned} < \text{ориент.переменная} > &::= < \text{ориентация} > \ : < \text{ид.} >, \\ < \text{ориентация} > &::= \textbf{in} \\ &::= \textbf{out} \end{aligned}$$

Будем говорить, что ориентированная переменная вида **in** v имеет входную ориентацию, а **out** v выходную. Идентификатор v в этом случае назовем идентификатором ориентированной переменной и обозначим $Id(\textbf{in}\ v) = Id(\textbf{out}\ v) = v$. Центральным в статической семантике языка является понятие схемы клоза — аналог типов данных в языках программирования:

$$< \text{схема клоза} > ::= < \text{посл.ор.перем.} > \ \vdash \ < \text{посл.ор.перем.} >,$$

где < посл.ор.перем. > — либо пустая строка, либо последовательность ориентированных переменных, разделенных запятыми. Последовательности переменных справа и слева от $\vdash$ назовем положительной и отрицательной частями схемы соответственно. Будем говорить, что вхождение ориентированной переменной слева от $\vdash$ имеет отрицательную полярность, а вхождение справа от $\vdash$ — положительную. Если cf — схема клоза, множество всех ориентированных переменных, имеющих отрицательную полярность в cf, обозначим $Neg(cf)$, а множество всех ориентированных переменных, имеющих положительную полярность в cf, —

$Pos(cf)$; $In(cf)$ и $Out(cf)$ обозначают множества идентификаторов всех ориентированных переменных, входящих в cf и имеющих входную и выходную ориентацию соответственно. $PI(cf), PO(cf), NI(cf), NO(cf)$ обозначают $\{v \mid \mathbf{in}\, v \in Pos(cf)\}$, $\{v \mid \mathbf{out}\, v \in \mathrm{Pos}(cf)\}$, $\{v \mid \mathbf{in}\, v \in \mathrm{Neg}(cf)\}$ и $\{v \mid \mathbf{out}\, v \in Neg(cf)\}$ соответственно. На схемы клозов накладывается дополнительное ограничение: если у двух различных вхождений ориентированных переменных в схеме клоза совпадают идентификаторы, то эти вхождения обязаны иметь различную ориентацию и различную полярность. Множество всех схем обозначим CF.

Пример. $\mathbf{in}\, X, \mathbf{out}\, Y \vdash \mathbf{out}\, X, \mathbf{in}\, Y$ — схема клоза.

В дальнейшем полагаем, что каждой стратегии (атомарной или построенной с помощью операций) сопоставляется некоторая схема клоза, которую назовем интерфейсом этой стратегии. Атомарные стратегии в ядре языка представляются обычными алфавитно-цифровыми идентификаторами:

$$< \text{ат.страт.} > ::= < \text{ид.} > .$$

Более сложные стратегии строятся с помощью операций пометки, перестановки, переименования, резолютивного замыкания и параллельной композиции; для группирования используются круглые скобки:

$$\begin{aligned} < \text{страт.} > \quad ::= \quad & < \text{ат.страт.} > \\ ::= \quad & (< \text{страт.} >) \\ ::= \quad & < \text{помеч.страт.} > \\ ::= \quad & < \text{перестановка} > \\ ::= \quad & < \text{переименование} > \\ ::= \quad & < \text{рез. зам.} > \\ ::= \quad & < \text{параллельная композиция} > . \end{aligned}$$

Определение. Будем говорить, что схемы клозов cf_1, cf_2 совместны, если любые два вхождения ориентированных переменных в cf_1 и cf_2 соответственно с одинаковым идентификатором либо имеют одинаковую полярность и одинаковую ориентацию, либо имеют разную полярность и разную ориентацию. Конечное множество схем назовем совместным, если все схемы, входящие в него, попарно совместны.

Примеры.

1. Схемы $\mathbf{in}\, X \vdash \mathbf{out}\, Y, \mathbf{out}\, Z$ и $\mathbf{in}\, Y \vdash \mathbf{out}\, X, \mathbf{out}\, Z$ совместны.
2. Схемы $\mathbf{in}\, X \vdash \mathbf{out}\, Y$ и $\mathbf{out}\, X \vdash \mathbf{out}\, Y$ несовместны.

Операция пометки имеет следующий синтаксис:

$$< \text{помеч.страт.} > ::= < \text{ид.} > : < \text{страт.} > .$$

Идентификатор, участвующий в записи, назовем меткой. Пометка не меняет интерфейса стратегии.

Операция перестановки имеет следующий синтаксис:

$$< \text{перестановка} > ::= \ [< \text{схема клоза} >] \ < \text{страт.} >,$$

где схема клоза, заключенная в квадратные скобки, совместна с интерфейсом стратегии, к которой применяется операция, и является интерфейсом результирующей стратегии.

Для описания синтаксиса переименования потребуется понятие формального аргумента. Элементарное переименование — это либо символ $\perp$, либо последовательность идентификаторов (без повторений), разделенных символом "|". Синтаксическая категория формальных аргументов описывается следующим правилом:

$$< \text{форм.арг.} > ::= < \text{послед.элем.пер.} > \vdash < \text{послед.элем.пер.} >,$$

где $<$ послед.элем.пер. $>$ — либо пустая строка, либо последовательность элементарных переименований, разделенных запятыми. Последовательности элементарных переименований справа и слева от $\vdash$ назовем соответственно положительной и отрицательной частями формального аргумента.

Определение. Пусть $cf = x_1, ..., x_k \ \vdash \ x_{k+1}, ..., x_l$ — схема клоза, $arg = y_1, ..., y_k \ \vdash \ y_{k+1}, ..., y_l$ — формальный аргумент.

Будем говорить, что arg соответствует cf, если выполнены следующие условия:

– для любых $i, j, i \neq j$, если x_i, x_j имеют одинаковую ориентацию, то элементарные переименования y_i, y_j не содержат общих идентификаторов;

– для любого i, если $x_i =$ **out** v, то y_i содержит не более одного идентификатора;

– найдется схема клоза cf' такая, что

$$\begin{aligned} NI(cf') &= \{\textbf{in } v | \text{для некоторого } i \in \{1, ..., k\} \ v \text{ входит в } y_i \\ &\quad \text{и } x_i \text{ имеет входную ориентацию}\}, \\ NO(cf') &= \{\textbf{out } v | \text{для некоторого } i \in \{1, ..., k\} \ v \text{ входит в } y_i \end{aligned}$$

$$
\begin{aligned}
& \text{и } x_i \text{ имеет выходную ориентацию}\},\\
PI(cf') &= \{\textbf{in } v|\text{для некоторого } i \in \{k+1,...,l\}\ v \text{ входит в } y_i\\
& \text{и } x_i \text{ имеет входную ориентацию}\},\\
PO(cf') &= \{\textbf{out } v|\text{для некоторого } i \in \{k+1,...,l\}\ v \text{ входит в } y_i\\
& \text{и } x_i \text{ имеет выходную ориентацию}\}.
\end{aligned}
$$

Будем говорить, что схема cf' порождается сравнением arg и cf. Сравнение формального аргумента со схемой клоза может порождать более одной схемы. Канонической схемой, порождаемой сравнением arg и cf, назовем $ch(\{cf' \mid cf' \text{ порождается сравнением } arg \text{ и } cf\})$.

Операция переименования имеет синтаксис

$$< \text{переименование} > ::= \ \{ < \text{форм.арг.} > \} \ < \text{страт.} >,$$

где формальный аргумент соответствует интерфейсу стратегии, к которой применяется операция, а порождаемая их сравнением каноническая схема является интерфейсом результирующей стратегии.

Синтаксис резолютивного замыкания определяется следующим образом:

$$
\begin{aligned}
< \text{рез.зам.} > &::= \textbf{resolve } < \text{страт.} > \textbf{ by } < \text{дир.предп.} >,\\
< \text{дир.предп.} > &::= \textbf{all } < \text{ид.} >\\
&::= \textbf{some } < \text{ид.} >\\
&::= < \text{дир.предп.} > \textbf{ or } < \text{дир.предп.} >\\
&::= < \text{дир.предп.} > \textbf{ orelse } < \text{дир.предп.} >\\
&::= (< \text{дир.предп.} >).
\end{aligned}
$$

Элементы синтаксической категории < дир.предп. > назовем директивами предпочтения. Множество всех директив предпочтения обозначим PD. Если pd — директива предпочтения, то $Vars(pd)$ обозначает множество всех идентификаторов, входящих в pd. Если cf — интерфейс стратегии s, то для того чтобы **resolve** s **by** pd было корректным описанием стратегии, потребуем $Vars(pd) \subseteq In(cf) \cap Out(cf)$. Интерфейс такого резолютивного замыкания получается из cf выбрасыванием всех вхождений ориентированных переменных со входной ориентацией и идентификатором из $Vars(pd)$.

Для определения синтаксиса параллельной композиции необходимо понятие надсхемы.

Определение. Пусть $\{cf_1, ..., cf_n\}$ — совместное множество схем. Любую схему cf, для которой

$$Neg(cf) = \bigcup_{i=1}^{n} Neg(cf_i)$$

и

$$Pos(cf) = \bigcup_{i=1}^{n} Pos(cf_i),$$

будем называть надсхемой для $cf_1, ..., cf_n$. Множество

$$SCF(\{cf_1, ..., cf_n\})$$

всех надсхем для $cf_1, ..., cf_n$ может содержать более одной схемы. Канонической надсхемой для $cf_1, ..., cf_n$ назовем

$$ch(SCF(\{cf_1, ..., cf_n\})).$$

Элемент синтаксической категории < парал.комп. > — это последовательность стратегий, разделенных символом "||". Множество интерфейсов всех стратегий в параллельной композиции должно быть совместно, а их каноническая надсхема является интерфейсом результирующей стратегии.

1.3. Декларативная семантика

1.3.1. Семантическая область

Приступая к описанию семантики языка стратегий, зафиксируем некоторое множество $Form$, которое назовем объектной областью. $Form$ можно считать мультимножеством (свободных) синтаксических объектов той логики, вывод в которой мы хотим автоматизировать (чаще всего формулы или объекты метасинтаксиса со свободными переменными). Элементы $Form$ для простоты назовем формулами. Зафиксируем также множество подстановок $Subst$ и функцию

$$apply : Form \times Subst \to Form$$

такую, что для любых $\varphi, \psi \in Form$, $\theta \in Subst$ из $\varphi \neq \psi$ следует $apply(\varphi, \theta) \neq apply(\psi, \theta)$. В дальнейшем для краткости вместо

$$apply(\varphi, \theta)$$

будем писать просто $\varphi\theta$. Полагаем что на $Subst$ определена бинарная операция $\circ$ такая, что для любых $\theta, \sigma \in Subst$, $\varphi \in Form$ имеет место $\varphi(\theta \circ \sigma) = (\varphi\theta)\sigma$. Кроме того, зафиксируем множество Val, элементы которого называются подтверждениями.

Определение. Тройку $< cf, input, output >$, где

$$cf \in \mathcal{CF},$$
$$input : In(cf) \to Mult(Mult^{+}(Form) \times Subst),$$
$$output : Out(cf) \to Mult(Form),$$

назовем клозом со структурой cf. Множество всех таких клозов обозначим $Cl(cf)$. Для любых $v \in In(cf), u \in Out(cf)$ мультимножество $input(v)$ называется входным местом клоза, а $output(u)$ — выходным местом.

Определение. Пару $< cl, val, cp >$, где $cl \in Cl(cf)$, $val \in Val$, $cp \in Seq(V)$, назовем выводом с корневым клозом cl, подтверждением val и путем вызова cp. Множество таких выводов обозначим $Inf(cf)$. Полагаем $Inf = \bigcup_{cf \in CF} Inf(cf)$.

С декларативной точки зрения каждой стратегии s с интерфейсом cf в качестве семантики приписывается некоторое подмножество $Inf(cf)$, которое обозначим $Sem(s)$. Интуитивно $Sem(s)$ означает множество выводов, которые могут быть получены стратегией s. Тактикам семантика приписывается произвольно с тем лишь условием, что все выводы, возвращаемые атомарной стратегией в качестве пути вызова, имеют пустую последовательность. Далее будет описана наша базовая дедуктивная система и в ее терминах мы определим семантику операций языка описания стратегий.

1.3.2. Базовая дедуктивная система

Правило преобразования схем.

Определение. Пусть cf, cf' — схемы. Будем говорить, что пара $< \varepsilon, \omega >$, где $\varepsilon \subseteq In(cf) \times In(cf')$, $\omega \subseteq Out(cf) \times Out(cf')$, является преобразователем cf в cf', при выполнении следующих условий:

1) если $< v, v' > \in \varepsilon$, то **in** v и **in** v' входят в cf и cf' с одинаковой полярностью;

2) если $< v, v' > \in \omega$, то **out** v и **out** v' входят в cf и cf' с одинаковой полярностью.

Будем считать, что фиксирована некоторая функция

$$tval : CF \times CF \times Inf \to Val.$$

Определение. Пусть $tr =< \varepsilon, \omega >$ — преобразователь схемы cf в схему cf'.

1. Рассмотрим клоз $cl =< cf, input, output >$. Если для любого $v \in Out(cf)$ из $\{v' | < v, v' > \in \omega\} = \emptyset$ следует $output(v) = \emptyset$, то полагаем $CT(tr, cl)$ равным множеству всех клозов вида $< cf', input', output' >$, удовлетворяющих следующим условиям:

– $input(v) = \bigcup_{<v,v'>\in\varepsilon} input'(v')$ для любого $v \in In(cf)$;

– если $< v, v' > \in \omega$, то $output'(v') = output(v)$.

В противном случае полагаем $CT(tr, cl) = \emptyset$.

2. Если $inf \in Inf(cf)$ — вывод с корневым клозом cl и путем вызова cp, то полагаем

$$IT(tr, inf) = \{< cl', tval(cf, cf', inf), cp > | cl' \in CT(tr, cl)\}.$$

Правило преобразования схем выглядит следующим образом:

$$\frac{inf}{inf'},$$

где $inf \in Inf(cf)$, tr — преобразователь схемы cf в некоторую схему cf' и $inf' \in IT(tr, inf)$.

Правило резолюции.

Определение. Определим функцию

$$PDI : PD \times Inf \to Mult(V \times Mult(Form))$$

следующим образом:

Пусть inf — вывод с корневым клозом $< cf, input, output >$.

1) $PDI(\textbf{all}\ v, inf) = \begin{cases} \{< v, output(v) >\} \text{ при } output(v) \neq \emptyset, \\ \emptyset \quad \text{в противном случае.} \end{cases}$

2) $PDI(\textbf{some}\ v, inf) = \begin{cases} \{< v, \{ch(output(v))\} >\} \text{ при } output(v) \neq \emptyset, \\ \emptyset \quad \text{в противном случае.} \end{cases}$

3) Если $pd_1, pd_2 \in PD$, то полагаем

$$PDI(pd_1\ \textbf{or}\ pd_2, inf) = PDI(pd_1, inf) \cup PDI(pd_2, inf).$$

4) Пусть $pd_1, pd_2 \in PD$.

$$PDI(pd_1 \textbf{ orelse } pd_2) = \begin{cases} PDI(pd_1, inf), \text{ если } PDI(pd_1, inf) \neq \emptyset, \\ PDI(pd_2, inf) \text{ в противном случае.} \end{cases}$$

Везде в дальнейшем полагаем, что фиксированы некоторые функции

$$rval : Inf \times Fin(Inf) \to Val$$

и

$$sval : \ Inf \times Subst \to Val.$$

Рассмотрим вывод inf с корневым клозом $< cf, input, output >$ и путем вывода cp. Пусть $\theta \in Subst$. Полагаем

$$SubstInf(inf, \theta) =<< cf, input', output' >, sval(inf, \theta), cp >,$$

где

$$input'(v) = \{< \Phi, \sigma \circ \theta > | < \Phi, \sigma > \in input(v)\}$$

для любого $v \in In(cf)$ и

$$output'(v) = \{\varphi\theta | \varphi \in output(v)\}$$

для любого $v \in Out(cf)$.

Определение. Рассмотрим директиву предпочтения pd и вывод inf с корневым клозом $< cf, input, output >$. Построим схему клоза cf' вычеркиванием из cf всех вхождений ориентированных переменных со входной ориентацией и идентификатором из $Vars(pd)$. Пусть для некоторого $n \geq 0$ $\{< v_1, \Phi_1, \theta_1 >, ..., < v_n, \Phi_n, \theta_n >\}$ — множество всех троек $< v, \Phi, \theta >$ таких, что $< \Phi, \theta > \in input(v)$, $v \in Vars(pd)$. Если $n > 0$ рассмотрим также мультимножество выводов $\{inf_1, ..., inf_n\}$ с корневыми клозами $< cf', input_i, output_i >$ такими, что для всякого $i \in \{1, ..., n\}$ $PDI(pd, inf_i)$ содержит пару $< v_i, \Psi_i >$ такую, что $\Phi_i \subseteq \Psi_i$. Для каждого $i \in \{1, ..., n\}$ полагаем $inf_i' = SubstInf(inf_i, \theta_i)$, а $< cf', input_i', output_i' >$ обозначим корневой клоз inf_i'. Определим $input'$, $output'$ следующим образом:

$$input'(v) = (\bigcup_{i=1}^{n} input_i'(v)) \cup input(v) \text{ для } v \in In(cf'),$$

$$output'(v) = output(v) \cup (\bigcup_{i=1}^{n} \Omega_i(v)) \text{ для } v \in Out(cf'),$$

где

$$\Omega_i(v) = \begin{cases} output'_i(v) \backslash \{\varphi\theta_i | \varphi \in \Phi_i\} \text{ при } v = v_i, \\ output'_i(v) \text{ в противном случае.} \end{cases}$$

Вывод

$$<< cf', input', output' >, rval(inf, \{inf'_1, ..., inf'_n\}), \lambda >$$

назовем резольвентой вывода inf с выводами $inf_1, ..., inf_n$ относительно директивы предпочтения pd и обозначим

$$Res(pd, inf, \{inf_i | 1 \leq i \leq n\}).$$

В случае $n = 0$ полагаем

$$Res(pd, inf, \emptyset) = inf.$$

Правило резолюции будет выглядеть следующим образом:

$$\frac{inf \qquad inf_1 \quad \ldots \quad inf_n}{Res(pd, inf, \{inf_i | 1 \leq i \leq n\})}.$$

1.3.3. Семантика пометки

Пусть s — стратегия, lab — идентификатор метки. Тогда

$$Sem(lab{:}s) = \{inf' | < cl, val, cp > \in Sem(s), < cl, val, lab.cp >\}.$$

1.3.4. Семантика переименования и перестановки

Определение. Пусть cf, cf' — совместные схемы. Определим

$$\varepsilon = \{< v, v > | v \in In(cf), v \in In(cf')\},$$

$$\omega = \{< v, v > | v \in Out(cf), v \in Out(cf')\}.$$

Тогда $< \varepsilon, \omega >$ — преобразователь cf в cf', который обозначим

$$PT(cf, cf').$$

Определение. Пусть $cf = x_1, ..., x_k \vdash x_{k+1}, ..., x_l$ — схема клоза, $arg = y_1, ..., y_k \vdash y_{k+1}, ..., y_l$ — формальный аргумент, соответствующий

cf, а cf' — каноническая схема, порождаемая их сравнением. Определим

$$\varepsilon = \{< v, v' > | \text{для некоторого } i \in \{1, ..., l\}\ x_i = \textbf{in}\ v, v' \text{ входит в } y_i\},$$

$$\omega = \{< v, v' > | \text{для некоторого } i \in \{1, ..., l\}\ x_i = \textbf{out}\ v, v' \text{входит в } y_i\}.$$

Тогда $< \varepsilon, \omega >$ — преобразователь cf в cf', который обозначим

$$RT(cf, cf', arg).$$

Теперь можно определить семантику операций перестановки и переименования. Пусть s — стратегия с интерфейсом cf. Если $s' = [cf']\ s$, то полагаем

$$Sem(s') = \bigcup_{inf \in Sem(s)} IT(PT(cf, cf'), inf).$$

Если $s' = \{arg\}\ s$, где arg соответствует cf, а cf' — каноническая схема, порождаемая их сравнением, то полагаем

$$Sem(s') = \bigcup_{inf \in Sem(s)} IT(RT(cf, cf'), inf).$$

1.3.5. Семантика параллельной композиции

Пусть $s' = (s_1||...||s_n), cf_i$ — интерфейс s_i а cf' — каноническая надсхема для $s_1, ..., s_n$. Полагаем $s'_i = [cf']\ s_i$. В качестве семантики параллельной композиции берется объединение

$$Sem(s') = \bigcup_{i=1}^{n} Sem(s'_i).$$

1.3.6. Семантика резолютивного замыкания

Определение. Рассмотрим $pd \in PD$ и множество выводов $I \subseteq Inf(cf)$. Множества $R_i, i \geq 0$, будем строить следующим образом: полагаем R_0 состоящим из всех резольвент вида $Res(pd, inf, \emptyset)$, где $inf \in I$ — вывод с корневым клозом $< cf, input, output >$ таким, что $input(v) = \emptyset$ для любого $v \in Vars(cf)$; R_{i+1} получается добавлением к R_i всех резольвент вида $Res(pd, inf, S)$, где $inf \in I$ и $S \subseteq R_i$. Множество $RC(I, pd) = \bigcup_{i=1}^{\infty} R_i$ назовем резолютивным замыканием I относительно pd. Полагаем

$$Sem(\textbf{resolve}\ s\ \textbf{by}\ pd) = RC(Sem(s), pd).$$

1.3.7. Связь между семантикой языка описания стратегий и реализуемыми логиками

Предположим, мы хотим автоматизировать поиск выводов в некоторой логике L. Чтобы сделать это корректно, мы должны понимать, как соотносится семантика L с семантикой нашего языка описания стратегий. Говоря о логике L, имеем в виду некоторое множество $Prop(L)$ синтаксических объектов-утверждений, о которых можно ставить вопрос о доказуемости в L; доказуемость в L задается выделением множества $Prov(L) \subseteq Prop(L)$ доказуемых объектов. Реализуя логику L в рамках нашей системы, мы должны указать некоторые $Form$, $Subst$, $tval$, $sval$, $rval$ и отображение $I_L : Inf \to Prop(L)$, называемое интерпретацией выводов из Inf в логике L.

Определение. Будем говорить, что L допускает правило

$$\frac{inf_1 \quad \dots \quad inf_n}{inf}$$

относительно интерпретации I_L, если из доказуемости всех

$$I_L(inf_1), ..., I_L(inf_n)$$

следует доказуемость $I_L(inf)$.

Определение. Стратегию s назовем корректной в логике L относительно интерпретации I_L, если

$$I_L(inf) \in Prov(L) \text{ для любого } inf \in Sem(s)$$

.

Утверждение. Если I_L такова, что L допускает правила нашей базовой дедуктивной системы (преобразование схем и резолюцию) и все атомарные стратегии корректны в L относительно I_L, то и любая стратегия, построенная в рамках нашего языка описания стратегий, корректна в L относительно интерпретации I_L.

Доказательство опускается.

1.4. Операционная семантика

1.4.1. Процессы, реализующие стратегии

Опишем (неформально) операционную семантику языка описания стратегий, лежащую в основе нашей реализации. С операционной точки зрения стратегии, описываемые средствами нашего языка, — это процессы, взаимодействующие некоторым образом со своим окружением. Пусть $Form, Subst$ и $apply$ такие же, как и в случае декларативной семантики (см. п.2.2.1). Пусть s — некоторая стратегия с интерфейсом cf. P_s будет обозначать соответствующий ей процесс. Взаимодействие P_s с окружением происходит посредством чтения данных из входных каналов и посылки результатов их обработки через единственный выходной канал. P_s имеет входные каналы $InCh(P_s, v)$ для всех переменных $v \in In(cf)$, по которым он может получать конечные непустые мультимножества синтаксических объектов из $Form$, и выходной канал $OutCh(P_s)$, по которому он может передавать построенные выводы. Заметим, что чтение данных через любые каналы $InCh(P_s, v)$, $v \in In(cf)$, и возврат выводов через $OutCh(P_s)$ может производиться параллельно и независимо. Состояние процесса P_s частично характеризуется множествами $InHist(P_s, v)$ конечных мультимножеств над $Form$, которые содержат все мультимножества, считанные процессом P_s через канал $InCh(P_s, v)$ на текущий момент, а также множеством $OutHist(P_s)$, содержащим все выводы, построенные процессом P_s и переданные им через канал $OutCh(P_s)$. При инициализации процесса P_s $OutHist(P_s) = \emptyset$ и $InHist(P_s, v) = \emptyset$ для всех $v \in In(cf)$. Каждый вывод inf, возвращаемый процессом P_s, обязан иметь в качестве корневого какой-либо клоз вида $< cf, input, output >$, для которого выполнено следующее условие: для любого $v \in In(cf)$ и любой пары $< \Phi, \theta > \in input(v)$ некоторое мультимножество $query(\Phi) \in InHist(P_s, v)$ содержит Φ. Тактикой может быть любой процесс, удовлетворяющий этому ограничению и строящий выводы с пустыми путями вызова. Покажем, как устроены процессы, соответствующие стратегиям, построенным с помощью операций.

1.4.2. Семантика пометки

По процессу P_s построим процесс $P_{lab:s}$, который функционирует следующим образом:

1. При инициализации $P_{lab:s}$ инициализируется P_s.

2. При попытке чтения процессом P_s через входной канал

$$InCh(P_s, v)$$

происходит чтение через

$$InCh(P_{lab:s}, v).$$

3. При попытке возврата процессом P_s некоторого вывода

$$< cl, val, cp >$$

производится возврат вывода

$$< cl, val, lab.cp > .$$

1.4.3. Семантика перестановки и переименования

Пусть s — стратегия с интерфейсом cf, а $tr =< \varepsilon, \omega >$ — преобразователь схемы cf в cf'. По процессу P_s построим процесс $PrT(tr, P_s)$, который функционирует следующим образом:

1. При инициализации $PrT(tr, P_s)$ инициализируется P_s.

2. При попытке чтения процессом P_s через входной канал

$$InCh(P_s, v)$$

процесс $PrT(tr, P_s)$ пытается читать данные из какого-либо канала

$$InCh(PrT(tr, P_s), v'),$$

где $v' \in In(cf')$ и $< v, v' > \in \varepsilon$, и передает их процессу P_s. Чтение процессом P_s через $InCh(P_s, v)$ блокируется до момента совершения чтения процессом $PrT(tr, P_s)$. При выборе

$$InCh(PrT(tr, P_s), v')$$

предпочтение отдается тем каналам, чтение из которых становится возможно раньше, чем из других каналов, если возможность чтения появляется одновременно у нескольких каналов, выбор производится недетерминированно.

3. При попытке возврата процессом P_s некоторого вывода

$$inf =<< cf, input, output >, val, cp >$$

сначала проверяется следующее условие: для любого $v \in Out(cf)$ такого, что $\{v' | < v, v' > \in \omega\} = \emptyset$ выполняется $output(v) = \emptyset$. Если это условие не выполняется, то возврат процессом P_s вывода считается совершенным. В противном случае строится вывод inf' с подтверждением $tval(cf, cf', inf)$, корневым клозом $< cf', input', output' >$ и путем вызова cp, где

$$input'(v') = \{< \Phi, \theta > | < \Phi, \theta > \in input(v),$$
$$< v, v' > \in \varepsilon,$$
$$query(\Phi) \in InHist(PrT(tr, P_s), v')\} \text{ для всех } v' \in In(cf'),$$
$$output'(v') = \bigcup_{< v, v' > \in \omega} output(v) \text{ для всех } v' \in Out(cf').$$

После этого процесс $PrT(tr, P_s)$ пытается вернуть inf' своему окружению. Возврат процессом P_s вывода inf считается совершенным с того момента, когда совершается возврат inf'.

Теперь мы готовы к описанию семантики операций перестановки и переименования. Пусть s — стратегия с интерфейсом cf. Если $s' = [cf']\ s$, то полагаем

$$P_{s'} = PrT(PT(cf, cf'), P_s).$$

Если $s' = \{arg\}\ s$, где arg соответствует cf, а cf' — каноническая схема, порождаемая их сравнением, то полагаем

$$P_{s'} = PrT(RT(cf, cf'), P_s).$$

1.4.4. Семантика параллельной композиции

Пусть $s' = (s_1 \ || \ ... \ || \ s_n)$. Не ограничивая общности, полагаем, что каждая из стратегий s_i имеет интерфейс cf. По $P_{s_i}, 1 \leq i \leq n$, построим $P_{s'}$. Считаем, что входные каналы всех P_{s_i} буферизованы. Буфер канала $InCh(P_{s_i}, v)$ обозначим $buff(P_{s_i}, v)$.

1. При инициализации $P_{s'}$ инициализируются все P_{s_i} с пустыми буферами $buff(P_{s_i}, v)$.

2. При попытке чтения каким-либо процессом P_{s_i} через входной канал $InCh(P_{s_i}, v)$ сначала опрашивается буфер $buff(P_{s_i}, v)$. Если этот буфер не пуст, его головной элемент удаляется и передается процессу P_{s_i} в качестве результата операции чтения через $InCh(P_{s_i}, v)$. Предположим, что $buff(P_{s_i}, v)$ пуст. В этом случае процесс $P_{s'}$ читает данные из некоторого канала $InCh(P_{s'}, v)$ и передает их процессу P_s через

$InCh(P_{s_i}, v)$. Чтение процессом P_{s_i} через $InCh(P_{s_i}, v)$ блокируется до совершения чтения процессом $P_{s'}$. Кроме того, считанные данные помещаются во все буферы $buff(P_{s_j}, v)$, $j \neq i$.

3. При попытке возврата процессом P_{s_i} некоторого вывода происходит возврат этого вывода процессом $P_{s'}$ своему окружению. Возврат вывода процессом P_{s_i} происходит при совершении соответствующего возврата процессом $P_{s'}$.

1.4.5. Семантика резолютивного замыкания

Пусть s — стратегия с интерфейсом cf, pd — директива предпочтения, $s' =$ **resolve** s **by** pd. По P_s будем строить $P_{s'}$.

1. При инициализации $P_{s'}$ инициализируется P_s. Помимо обычных параметров состояние $P_{s'}$ будет характеризоваться функцией

$$Src: \bigcup_{v \in Vars(pd)} InHist(P_s, v) \to OutHist(P_{s'}).$$

2. При попытке чтения процессом P_s через входной канал

$$InCh(P_s, v),\ v \notin Vars(pd),$$

производится чтение через $InCh(P_{s'}, v)$, полученные данные передаются P_s. При попытке чтения через

$$InCh(P_s, v),\ v \in Vars(pd),$$

происходит передача данных процессом $P_{s'}$ процессу P_s через этот канал. Откуда берутся такие данные — будет ясно из следующего пункта.

3. При попытке возврата процессом P_s некоторого вывода

$$inf =<< cf, input, output >, val, cp >$$

производятся следующие действия:

– Пусть

$$\{\Phi_1, ..., \Phi_n\} = \{\Phi | < \Phi, \theta > \in input(v), v \in Vars(pd)\}.$$

Для $i \in \{1, ..., n\}$ полагаем $inf_i = Src(query(\Phi_i))$. Строим резольвенту

$$inf' = Res(pd, inf, \{inf_1, ..., inf_n\}),$$

после чего процесс $P_{s'}$ пытается вернуть ее своему окружению. Как только такой возврат совершается, переходим к следующему шагу.

– Вычисляем интерпретацию директивы предпочтения pd на выводе inf'. Пусть

$$PDI(pd, inf') = \{< v_1, \Psi_1 >, ..., < v_m, \Psi_m >\}.$$

Если $m = 0$, то считаем возврат процессом P_s вывода inf совершенным. Если $m > 0$, то процесс $P_{s'}$ пытается передать процессу P_s некоторое мультимножество Ψ_i через канал $InCh(P_s, v_i)$. При выборе предпочтение отдается тем каналам, передача по которым становится возможна раньше, чем из других каналов, если возможность передачи появляется одновременно у нескольких каналов, выбор производится недетерминированно. После того как происходит передача некоторого Ψ_i, возврат процессом P_s считается совершенным, а Src доопределяется на Ψ_i следующим образом: $Src(\Psi_i) = inf'$.

1.4.6. Тупиковые ситуации (дедлоки)

Заметим, что при таком определении операционной семантики наши стратегии могут в некоторых случаях приходить в состояние дедлока. Рассмотрим следующий пример. Возьмем стратегию

$s =$ [**in** $x \vdash$ **out** y]
resolve
([**in** $x \vdash$ **out** y] s_1 ||
[**in** $y \vdash$ **out** x] s_2)
by some y **or some** x

Предположим, стратегии s_1 и s_2 реализованы таким образом, что при блокировании возврата некоторого вывода становится невозможным чтение через входной канал. Может возникнуть ситуация, когда при блокированном возврате вывода стратегией s_1 стратегия s_2 также пытается совершить возврат, что порождает дедлок. В реализации системы эта проблема частично решается введением средств явной буферизации как входных, так и выходных каналов.

1.5. Примеры

Проиллюстрируем возможности нашего языка описания стратегий несколькими простыми примерами. Предполагаем, что фиксирована объектная область $Form$ и множество подстановок $Subst$. Предварительно

условимся об обозначениях: ϵ обозначает такую подстановку, что для любой формулы $\varphi \in Form$ и любой подстановки $\theta \in Subst$ имеет место $\varphi\epsilon = \varphi$ и $\theta \circ \epsilon = \epsilon \circ \theta = \theta$. Предположим, что $cf = x_1, ..., x_m \vdash x_{m+1}, ..., x_n$ — некоторая схема клоза. Тогда клоз $< cf, input, output >$ запишем следующим образом:

$$x_1 = \Phi_1, ..., x_m = \Phi_m \vdash x_{m+1} = \Phi_{m+1}, ..., x_n = \Phi_n,$$

где $\Phi_i = input(v_i)$, если $x_i = \textbf{in}\ v_i$, и $\Phi_i = output(v_i)$, если $x_i = \textbf{out}\ v_i$. Пример записи клоза:

$$\textbf{out}\ Subgoals = \{\varphi, \psi\} \vdash \textbf{in}\ Goal = \{< \{\varphi \wedge \psi\}, \epsilon >\}.$$

Природа подтверждений в выводах нас интересовать не будет, поэтому каждый вывод будем идентифицировать с его корневым клозом. Например, говоря о семантике стратегии, будем указывать не множество выводов, а множество соответствующих клозов.

Пример 1. Последовательная композиция дедуктивных процедур.

Пусть существуют две дедуктивные процедуры P_1 и P_2, каждая из которых берет некоторую формулу φ и строит по ней какое-то множество $P_i(\varphi)$ (возможно бесконечное) формул-следствий. Требуется построить процедуру P, которая по формуле φ строит

$$P(\varphi) = \bigcup_{\psi \in P_1(\varphi)} P_2(\psi).$$

Считаем что P_1 и P_2 реализованы в виде стратегий $Strat_1, Strat_2$, каждая из которых имеет интерфейс

$$\textbf{in}\ Form\ \vdash\ \textbf{out}\ Conseq;$$

$Sem(Strat_i)$ состоит из всех клозов вида

$$\textbf{in}\ Form = \{< \{\varphi\}, \epsilon >\} \vdash \textbf{out}\ Conseq = \{\psi\},$$

где φ — произвольная формула, ψ — следствие из φ.

Процедура P реализуется следующей стратегией:

[**in** $Form\ \vdash$ **out** $Conseq$]
resolve
($\{Form\ \vdash\ X\}\ Strat_1\ ||$

$\{X \vdash Conseq\}\ Strat_2)$
by all X

Пример 2. Условное применение процедур.

Пусть P_1, P_2 — процедуры из предыдущего примера. Пусть C — некоторое условие, эффективно проверяемое на формулах из $Form$. Нам хотелось бы построить процедуру P_{C}, которая по формуле φ строит

$$P_C = \begin{cases} P_1(\varphi), \text{ если условие } C \text{ истинно на } \varphi, \\ P_2(\varphi) \text{ если условие } C \text{ ложно на } \varphi. \end{cases}$$

Заведем атомарную стратегию $Cond_C$ с интерфейсом

$$\textbf{in}\ Form \vdash \textbf{out}\ A, \textbf{out}\ B$$

и со следующей семантикой: $Sem(Cond_C)$ — множество, состоящее из всех клозов вида

$$\textbf{in}\ Form = \{<\varphi, \epsilon>\} \vdash \textbf{out}\ A = \{\varphi\}, \textbf{out}\ B = \emptyset,$$

где $\varphi \in Form$ и C истинно на φ, и всех клозов вида

$$\textbf{in}\ Form = \{<\varphi, \epsilon>\} \vdash \textbf{out}\ A = \emptyset, \textbf{out}\ B = \{\varphi\},$$

где $\varphi \in Form$ и C ложно на φ.

Искомая процедура P_C может быть реализована следующей стратегией:

[**in** $Form \vdash$ **out** $Conseq$]
resolve
$(\{Form \vdash X, Y\}\ Cond_C \ ||$
$\{X \vdash Conseq\}\ Strat_1 \ ||$
$\{Y \vdash Conseq\}\ Strat_2)$
by all X **or all** Y

Пример 3. Итерация.

Пусть P_1 — процедура из примера 1. Пусть φ — формула. Полагаем для каждого $\varphi \in Form$

$$P_1^0(\varphi) = P_1(\varphi),$$
$$P_1^{i+1}(\varphi) = \bigcup_{\psi \in P_1^i(\varphi)} P_1(\psi).$$

Предположим, нам хотелось бы иметь процедуру P, которая по формуле φ строит замыкание

$$P(\varphi) = \bigcup_{i=0}^{\infty} P_1^i(\varphi).$$

Такую процедуру можно реализовать следующей стратегией:

[**in** $Form \vdash$ **out** $Conseq$]
resolve ($\{Form \mid Conseq \vdash Conseq\}$ $Strat_1$) **by all** $Conseq$

Заметим, что если стратегия $Strat_1$ не может при блокированной попытке возврата считывать новые формулы через канал, помеченный $Form$, то она блокируется после возврата первого же вывода (см. п. 2.3.6).

Пример 4. Использование лемм для управления поиском вывода.

При доказательстве сложных утверждений практически невозможно обойтись без планирования поиска вывода посредством указания промежуточных лемм. Пусть $T1, T2$ — два множества замкнутых формул. Предположим, мы хотим доказать, что из конъюнкции формул T_1, T_2 следует некоторая целевая замкнутая формула G. Предположим, что у нас имеются две стратегии $Strat_1$ и $Strat_2$ с интерфейсом

$$\textbf{in}\ Theory \vdash \textbf{in}\ Goal.$$

Из каких-либо соображений можно указать лемму L и попытаться доказать $T_1, T_2 \vdash G$, доказывая $T_1 \vdash L$ с помощью $Strat_1$ и $T_2, L \vdash G$ с помощью $Strat_2$. Для этого заводим атомарную стратегию $Lemma_L$ с интерфейсом

$$\textbf{out}\ Prem \vdash \textbf{out}\ Conc,$$

семантика которой состоит из одного клоза

$$\textbf{out}\ Prem = \{L\} \vdash \textbf{out}\ Conc = \{L\}.$$

Для решения задачи можно использовать следующую стратегию:

[**in** T_1, **in** $T_2 \vdash$ **in** $Goal$]
resolve
($\{T_1 \vdash Lemma\}$ $Strat_1$ ||
$\{Lemma \vdash Lemma'\}$ $Lemma_L$ ||

$\{(T_2 \mid Lemma') \vdash Goal\}\ Strat_2)$
by some $Lemma$ **or some** $Lemma'$

Можно фиксировать порядок применения стратегий $Strat_1$ и $Strat_2$, заменяя директиву предпочтения на

some $Lemma$ **orelse some** $Lemma'$

или

some $Lemma'$ **orelse some** $Lemma$.

Кроме того, в некоторых случаях может оказаться желательным параллельное выполнение $Strat_1$ и $Strat_2$, для чего заведем атомарную стратегию $Unify$ с интерфейсом

in $Prem \vdash$ **in** $Conc$

и семантикой, состоящей из всех клозов вида

in $Prem = \{< \{\varphi_1\}, \theta_1 >\} \vdash$ **out** $Conc = \{< \{\varphi_2\}, \theta_2 >\}$,

где формула $\varphi_1\theta_1$ эквивалентна (в реализуемой логике) формуле $\varphi_2\theta_2$. Искомая стратегия может выглядеть следующим образом:

[**in** T_1, **in** $T_2 \vdash$ **in** $Goal$]
resolve
($\{T_1 \vdash Lemma_1\}\ Strat_1$ ||
$\{Lemma1 \vdash Lemma'\}\ Lemma_L$ ||
$\{Lemma' \vdash Lemma''\}\ Unify$ ||
$\{Lemma'' \vdash Lemma_2\}\ Lemma_L$ ||
$\{(T_2 \mid Lemma_2) \vdash Goal\}\ Strat_2$)
by (**some** $Lemma_1$ **or some** $Lemma_2$)
orelse (**some** $Lemma'$ **or some** $Lemma''$

Иногда явный вид леммы можно не указывать, достаточно указать способ, которым леммы получаются и/или применяются. Пусть теперь $Strat_1, Strat_2$ имеют интерфейсы

in $Theory \vdash$ **out** $Conseq$

и

out $ProofOblig \vdash$ **in** $Goal$

соответственно. Пусть стратегия $Solver$ с интерфейсом

$$\textbf{in}\ Theory\ \vdash\ \textbf{in}\ Goal$$

реализует некоторую разрешающую процедуру. Для решения задачи может подойти следующая стратегия:

$[\textbf{in}\ Theory_1, \textbf{in}\ Theory_2\ \vdash\ \textbf{in}\ Goal]$
resolve
$(\{Theory_1\ \vdash\ X\}\ Strat_1\ ||$
$\{X\ \vdash\ Y\}\ Solver\ ||$
$\{Y\ \vdash\ Goal\}\ Strat_2)$
by all X **or all** Y

Пример 5. Полуразрешающая процедура для логики предикатов первого порядка (на основе гиперрезолюции).

В рамках данного примера предполагаем, что реализуемой является логика предикатов первого порядка (ЛППП). Объектной области $Form$ соответствует множество формул ЛППП со свободными переменными. Условимся об обозначениях. Символами $\wedge, \vee, \neg, \Rightarrow$ обозначим пропозициональные связки: конъюнкцию, дизъюнкцию, отрицание и импликацию соответственно, а символами $\exists, \forall$ — экзистенциальный и универсальный кванторы. Связанные переменные обозначим алфавитно-цифровыми идентификаторами. Свободные переменные и скулемовские константы обозначим идентификаторами со знаками ? и ! на конце. Выводы интерпретируются следующим образом: если вывод inf имеет корневой клоз

$$cl =< cf, input, output >,$$

то полагаем

$$I_{\text{ЛППП}}(inf) = \forall\overline{x}.I^{out}(cl)\ \Rightarrow\ I^{in}(cl),$$

где

$$I^{out}(cl) = \bigwedge_{v\in NO(cf)}(\bigwedge\ output(v))\ \Rightarrow\ \bigvee_{v\in NO(cf)}(\bigvee\ output(v)),$$
$$I^{in}(cl) = \bigwedge_{v\in PI(cf)}\ F(v)\ \Rightarrow\ \bigvee_{v\in PI(cf)}\ G(v)),$$
$$F(v) = \bigwedge_{<\Phi,\theta>\in input(v)}(\bigvee\ \{\varphi\theta|\varphi\in\Phi\}),$$
$$G(v) = \bigvee_{<\Phi,\theta>\in input(v)}(\bigwedge\ \{\varphi\theta|\varphi\in\Phi\}),$$

$\overline{x}$ — все свободные переменные, входящие в формулу

$$I^{out}(cl)\ \Rightarrow\ I^{in}(cl).$$

Пусть теперь Γ — некоторое мультимножество формул. Необходимо реализовать процедуру, которая находила бы конечные мультимножества $\{\varphi_1, ..., \varphi_m\} \subseteq \Gamma$, $m \geq 0$, и подстановки $\theta_i, 1 \leq m$ такие, что формула $\forall \overline{x}.\psi$ являлась бы теоремой *ЛПППП*, где

$$\psi = \varphi_1\theta_1 \ \vee ... \vee \varphi_m\theta_m$$

и $\overline{x}$ — все свободные переменные в ψ. Для реализации такой процедуры потребуется несколько атомарных стратегий. Для построения ДНФ по некоторой формуле используем атомарные стратегии

$$Forward,\ Backward,\ FAtom,\ BAtom$$

с интерфейсами

$$\begin{array}{l}\textbf{out}\ Subgoals\ \vdash\ \textbf{in}\ Goal, \textbf{out}\ Alt,\\ \textbf{in}\ Assumption, \textbf{out}\ Cond\ \vdash\ \textbf{out}\ Coclusion,\\ \textbf{out}\ Atomary, \textbf{out}\ NonAtomary\ \vdash\ \textbf{in}\ Goal,\\ \textbf{in}\ Assum\ \vdash\ \textbf{out}\ Atomary, \textbf{out}\ NonAtomary\end{array}$$

соответственно. Семантика атомарной стратегии Forward будет состоять из всех клозов вида:

$$\begin{array}{l}\textbf{out}\ Subgoals = \{\varphi, \psi\}\ \vdash \textbf{in}\ Goal = \{< \{\varphi \wedge \psi\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\\ \textbf{out}\ Subgoals = \{\varphi\} \vdash \textbf{in}\ Goal = \{< \{\varphi \vee \psi\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\\ \textbf{out}\ Subgoals = \{\psi\} \vdash \textbf{in}\ Goal = \{< \{\varphi \vee \psi\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\\ \textbf{out}\ Subgoals = \{\psi\} \vdash \textbf{in}\ Goal = \{< \{\varphi \Rightarrow \psi\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\\ \textbf{out}\ Subgoals = \emptyset \vdash \textbf{in}\ Goal = \{< \{\varphi \Rightarrow \psi\}, \epsilon >\}, \textbf{out}\ Alt = \{\varphi\},\\ \textbf{out}\ Subgoals = \emptyset \vdash \textbf{in}\ Goal = \{< \{\neg\varphi\}, \epsilon >\}, \textbf{out}\ Alt = \{\varphi\},\\ \textbf{out}\ Subgoals = \{\varphi(x?)\} \vdash \textbf{in}\ Goal = \{< \{\exists x.\varphi(x)\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\\ \textbf{out}\ Subgoals = \{\varphi(x!(\overline{y}))\} \vdash\\ \quad \textbf{in}\ Goal = \{< \{\forall x.\varphi(x)\}, \epsilon >\}, \textbf{out}\ Alt = \emptyset,\end{array}$$

где $x!$ — скулемовская функциональная константа (новая), $\overline{y}$ — все свободные переменные, входящие в $\forall x.\varphi(x)$.

Семантика атомарной стратегии Backward будет состоять из всех

клозов вида:

$$\begin{array}{l}
\textbf{in } Assum = \{< \{\varphi \wedge \psi\}, \epsilon >\}, \textbf{out } Cond = \emptyset \vdash \textbf{out } Conc = \{\varphi\}, \\
\textbf{in } Assum = \{< \{\varphi \wedge \psi\}, \epsilon >\}, \textbf{out } Cond = \emptyset \vdash \textbf{out } Conc = \{\psi\}, \\
\textbf{in } Assum = \{< \{\varphi \vee \psi\}, \epsilon >\}, \textbf{out } Cond = \emptyset \vdash \textbf{out } Conc = \{\varphi, \psi\}, \\
\textbf{in } Assum = \{< \{\varphi \Rightarrow \psi\}, \epsilon >\}, \textbf{out } Cond = \varphi \vdash \textbf{out } Conc = \{\psi\}, \\
\textbf{in } Assum = \{< \{\neg\varphi\}, \epsilon >\}, \textbf{out } Cond = \{\varphi\} \vdash \textbf{out } Conc = \emptyset, \\
\textbf{in } Assum = \{< \{\forall x.\varphi(x)\}, \epsilon >\}, \textbf{out } Cond = \emptyset \vdash \textbf{out } Conc = \varphi(x?), \\
\textbf{in } Assum = \{< \{\exists x.\varphi(x)\}, \epsilon >\}, \textbf{out } Cond = \emptyset \vdash \\
\quad \textbf{out } Conc = \{\varphi(x!(\overline{x}))\},
\end{array}$$

где $x!$ — скулемовская функциональная константа (новая), $\overline{x}$ — все свободные переменные, входящие в $\forall x.\varphi(x)$.

Семантика атомарной стратегии FAtom будет состоять из всех клозов вида:

$$\textbf{out } Atomary = \Phi_1, \textbf{out } NonAtomary = \Phi_2 \vdash \textbf{in } Goal = \{< \Phi, \epsilon >\},$$

где $\Phi = \Phi_1 \cup \Phi_2$, все формулы из Φ_1 атомарные, а формулы из Φ_2 — неатомарные.

Семантика атомарной стратегии BAtom будет состоять из всех клозов вида:

$$\textbf{in } Assum = \{< \Phi, \epsilon >\} \vdash \textbf{out } Atomary = \Phi_1, \textbf{out } NonAtomary = \Phi_2,$$

где $\Phi = \Phi_1 \cup \Phi_2$, все формулы из Φ_1 атомарные, а формулы из Φ_2 — неатомарные.

Комбинируя перечисленные стратегии, построим стратегию DNF, равную

$$\begin{array}{l}
[\textbf{out } NegAtom \vdash \textbf{out } PosAtom, \textbf{in } Goal] \\
\quad \textbf{resolve} \\
\quad\quad (\{Subgoals \vdash Subgoals', Alt\}\ Forward\ || \\
\quad\quad\ \{NegAtom, Subgoals' \vdash (Goal|Subgoals|Cond)\}\ FAtom\ || \\
\quad\quad\ \{Conc', Cond \vdash Conc\}\ Backward\ || \\
\quad\quad\ \{(Alt|Conc) \vdash PosAtom, Conc'\}\ BAtom) \\
\quad \textbf{by some } Subgoals' \\
\quad\quad \textbf{or some } Conc' \\
\quad\quad \textbf{or all } Subgoals \\
\quad\quad \textbf{or all } Conc
\end{array}$$

Для построения полуразрешающей процедуры потребуются еще две атомарные стратегии — $Factor$ и $Unify$ с интерфейсами

$$\textbf{in}\ AtomSet\ \vdash\ \textbf{out}\ Factored$$

и

$$\textbf{in}\ PosAt\ \vdash\ \textbf{in}\ NegAt$$

соответственно.

Определение. Пусть Φ — конечное мультимножество атомарных формул.

1. Предположим, найдутся две унифицируемые формулы $\varphi_1, \varphi_2 \in \Phi$. Мультимножество $\{\psi\theta | \psi \in \Phi \setminus \{\varphi_2\}\}$ назовем бинарной склейкой Φ с подстановкой θ, если θ — наиболее общий унификатор φ_1 и φ_2.

2. Определим понятие склейки Φ и соответствующей ей подстановки.

– Само Φ является склейкой Φ с подстановкой ϵ.

– Любая бинарная склейка мультимножества Φ с подстановкой θ является склейкой Φ.

– Пусть Φ' — склейка Φ с подстановкой θ'. Тогда любая бинарная склейка мультимножества Φ' с подстановкой θ является склейкой Φ с подстановкой $\theta' \circ \theta$.

– Других склеек нет.

Семантика атомарной стратегии $Factor$ состоит из всех клозов вида:

$$\textbf{in}\ AtomSet = \{< \Phi, \theta >\}\ \vdash\ \textbf{out}\ Factored = \Psi,$$

где Φ, Ψ состоят только из атомарных формул, Ψ — склейка Φ с подстановкой θ.

Семантика атомарной стратегии $Unify$ будет состоять из всех клозов вида:

$$\textbf{in}\ PosAt = \{< \{\varphi\}, \theta_1 >\}\ \vdash\ \textbf{in}\ NegAt = \{< \{\psi\}, \theta_2 >\},$$

где $< \theta_1, \theta_2 >$ — наиболее общий унификатор атомарных формул φ и ψ.

Интересующую нас стратегию (положительную гиперрезолюцию) можно определить следующим образом:

$PosHypRes =$
[$\vdash$ **in** $Goal$]
resolve
($\{NegAtom\ \vdash\ PosAtom, Goal\}\ DNF\ ||$
$\{PosAtom\ \vdash\ PosAtom'\}\ Factor\ ||$

$\{PosAtom' \vdash NegAtom\}\ Unify)$
by some $NegAtom$ **orelse all** $PosAtom$ **orelse all** $PosAtom'$

Заметим, что данный пример достаточно исскуствен. Использование процедуры, определенной таким образом, в реальных доказательствах невыгодно из-за неэффективности. На практике было бы разумнее эффективно реализовать гиперрезолюцию (и многие другие универсальные и часто используемые дедуктивные процедуры) в виде атомарной стратегии. Преимущества нашего подхода — быстрое и наглядное описание стратегий — проявляются при "штучном"планировании решений для конкретных дедуктивных задач (или узких классов задач) и, возможно, при прототипировании некоторых универсальных процедур.

Пример 6. Комбинирование разрешающих процедур.

Как и в предыдущем примере, полагаем, что реализуемой логикой является *ЛППП*. Рассмотрим некоторую теорию T сигнатуры Σ, содержащей специальный 0-местный предикатный символ $true$, интерпретируемый как истинный в любой модели. Пусть P — упрощающая процедура, которая по произвольной формуле φ сигнатуры Σ строит формулу $P(\varphi)$ такую, что $\forall\overline{x}.(P(\varphi) \Leftrightarrow \varphi) \in T$, где $\overline{x}$ — все свободные переменные формулы $P(\varphi) \Leftrightarrow \varphi$. Назовем такую процедуру разрешающей для теории T, если для любой формулы φ $\exists\overline{x}\varphi \in T$ равносильно $P(\varphi) = true$, где $\overline{x}$ — все свободные переменные формулы φ. Пусть теперь T_1, T_2 — две теории сигнатуры Σ и P_1, P_2 — разрешающие процедуры для T_1, T_2 соответственно. Наша задача — построить разрешающую процедуру для дедуктивного замыкания $T_1 \cup T_2$. Для ее решения мы намерены реализовать обобщение метода Нельсона - Оппена (см. [8, 9]). Предполагаем, что процедуры P_1, P_2 реализованы в виде стратегий $Solver_1, Solver_2$ с интерфейсом

$$\textbf{out}\ Rejected, \textbf{out}\ Simplified \vdash\ \textbf{in}\ Goal$$

и семантикой, состоящей из всех клозов вида

$$\textbf{out}\ Rejected = \emptyset, \textbf{out}\ Simplified = \{\psi_1, .., \psi_n\} \vdash \textbf{in}\ Goal = \{< \{\varphi\}, \epsilon >\},$$

где $n \geq 0$, $\forall\overline{x}.\psi_1 \vee ... \vee \psi_n \Leftrightarrow \varphi \in T_i$, и клозов вида

$$\textbf{out}\ Rejected = \{\varphi\}, \textbf{out}\ Simplified = \emptyset \vdash \textbf{in}\ Goal = \{< \{\varphi\}, \epsilon >\}.$$

Клозы первого вида появляются тогда, когда формула φ "может быть упрощена"соответствующей процедурой P_i, причем случай $n = 0$

соответствует $P_i(\varphi) = true$. Клозы второго вида появляются, если процедура P_i "не применима"к φ. Сделанные предположения позволяют описать искомую стратегию следующим образом:

$Solver =$
  [**out** $Rejected$, **out** $Simplified \vdash$ **in** $Goal$]
    resolve
      ($\{R1, S1 \vdash Goal\}$ $Solver1$ ||
       $\{Rejected, S2 \vdash R1\}$ $Solver2$ ||
       $\{Simplified, S1 \vdash S2\}$ $Solver1$ ||
       $\{Simplified, S2 \vdash S1\}$ $Solver2$)
    by all $R1$ **orelse** (**some** $S1$ **or some** $S2$)

2. ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ СИСТЕМЫ

Излагаемый далее материал предполагает знакомство читателя с языком SML (включая систему модулей SML) и основными понятиями, используемыми при программировании на CML (легковесные процессы и CML-события).

2.1. Структура системы

Основная часть системы представляет собой три библиотечных модуля на языке CML:

– файл `BUCEPHALAS.sml`, содержащий описание сигнатуры `Bucephalas`;

– файл `Bucephalas.sml`, содержащий описание функтора `Bucephalas`, который по некоторой структуре с сигнатурой Logic строит структуру с сигнатурой `Bucephalas`;

– файл `LOGIC.sml`, содержащий описание сигнатуры `Logic`.

Для построения доказывателя для какой-либо логики на основе нашей системы нужно описать некоторую структуру L, удовлетворяющую сигнатуре `Logic`, которая должна содержать определения типов данных для представления синтаксических объектов реализуемой логики (формул, подстановок и доказательств) и операций над ними. После этого, применяя функтор `Bucephalas` к структуре L, строим структуру, содержащую определения типов данных и операций, необходимых для описания стратегий.

2.2. Сигнатура `Logic`: спецификация модуля, реализующего логику

Модуль, реализующий синтаксис объектной логики, должен содержать описания следующих типов данных:

– `form`, `subst` и `validation`, соответствующих множествам $Form$, $Subst$ и Val в формальной семантике языка описания стратегий;

– `freeVars`, служащий для представления конечных множеств свободных переменных, входящих в формулы;

– `stratInfo`, являющийся вспомогательным. Элементами этого типа могут помечаться некоторые стратегии, после чего каждый вывод, построенный такой стратегией, будет также помечен соответствующим значением. Примером использования `stratInfo` может являться именование стратегий: некоторым стратегиям присваиваются имена-строки, после чего выводы, построенные именованными стратегиями, ссылаются на соответствующие имена, что удобно при чтении построенных доказательств.

Функции

```
apply: form * subst -> form
```

и

```
comp: subst * subst -> subst
```

соответствуют операциям *apply* и $\circ$ в формальной семантике. Функция

```
freeVars:form -> freeVars
```

по формуле находит множество входящих в нее свободных переменных. Функция

```
freeVarsUnion: freeVars * freeVars -> freeVars
```

позволяет строить объединение двух множеств свободных переменных. Функция

```
newFreeVars: freeVars * subst -> freeVars
```

по формуле и подстановке позволяет определить множество свободных переменных, входящих в формулу, получающуюся применением подстановки к исходной формуле. Наличие двух последних операций необходимо для нахождения множеств свободных переменных, входящих в

клоз. Функции

```
tval: 'clauseFramework * 'clauseFramework * 'inference ->
       validation,
sval: subst * 'inference -> validation
rval: 'inference * ('inference list) -> validation
```

соответствуют функциям *tval*, *sval* и *rval*, используемым в формальной семантике. При применении этих функций для построения подтверждений в выводах типовым переменным

`'clauseFramework`

и

`'inference`

сопоставляются типы данных

`clauseFramework`

и

`inference`,

определяемые функтором `Bucephalas` для представления схем клозов и выводов. Содержимое файла `LOGIC.sml` приведено полностью в приложении.

2.3. Пользовательские функции системы: сигнатура `Bucephalas`

2.3.1. Реализуемая логика

Типы данных

`form`, `subst`, `validation`, `freeVars`, `stratInfo`

идентичны одноименным типам структуры, реализующей синтаксис логики.

2.3.2. Схемы клозов и формальные аргументы

Для представления ориентированных переменных служит тип

`orientVar`.

Идентификаторы ориентированных переменных представляются элементами встроенного типа

`string`.

Функции

```
entry: string -> orientVar
```

и

```
outcome: string -> orientVar
```

используются для создания ориентированных переменных со входной и выходной ориентацией соответственно, для распознавания которых применяются предикаты

```
isEntryVar: orientVar -> bool
```

и

```
isOutcomeVar: orientVar -> bool
```

соответственно. Для определения идентификатора ориентированной переменной используется функция

```
varId: orientVar -> string.
```

Тип данных

`clauseFramework`

служит для представления схем клозов. Функция

```
mkClauseFramework: orientVar list * orientVar list ->
                                        clauseFramework
```

используется для создания схем клозов. Если l_1, l_2 — списки ориентированных переменных, то `mkClauseFramework`(l_1, l_2) дает в результате схему клоза с положительной частью l_2 и отрицательной частью l_1. Если построение такой схемы невозможно, порождается исключение

`InvalidVariableOccurence`.

Функции

```
positive: clauseFramework -> orientVar list
```

и

```
negative: clauseFramework -> orientVar list
```

позволяют находить соответственно положительную и отрицательную части схемы клоза. Тип

```
formArg
```

служит для представления формальных аргументов, а для их создания используется функция

```
mkFormArg: (string list) list * (string list) list -> formArg.
```

Если l_1, l_2 — списки списков идентификаторов, представляющие последовательности элементарных переименований, то `mkFormArg`(l_1, l_2) дает в результате формальный аргумент с положительной частью l_2 и отрицательной частью l_1.

2.3.3. Клозы

Для представления клозов существует тип данных

```
clause.
```

Структура клоза находится с помощью функции

```
framework: clause -> clauseFramework.
```

Функции

```
input: clause -> string -> (form list * subst) list
```

и

```
output: clause -> string -> form list
```

используются для извлечения содержимого входных и выходных мест клоза. Если в качестве второго аргумента указана строка, не являющаяся идентификатором, входящим в структуру клоза с соответствующей ориентацией, то порождаются исключения

```
InvalidEntryId
```

и

```
InvalidOutcomeId
```

соответственно.

2.3.4. Выводы

Тип

`inference`

служит для представления выводов. Корневой клоз, подтверждение и путь вызова некоторого вывода вычисляются с помощью функций

```
root: inference -> clause,
validation: inference -> validation,
callPath: inference -> string list
```

соответственно. Если вывод помечен значением типа `stratInfo`, то это значение можно получить с помощью функции

`stratInfo: inference -> stratInfo option`.

2.3.5. Стратегии

Стратегии в нашей системе представляются элементами типа

`strategy`.

Интерфейс стратегии может быть найден с помощью функции

`interface: strategy -> clauseFramework`.

Инициализировать стратегию в окружении, которое не передает ей никаких данных через входные каналы, можно с помощью функции

`run: strategy -> inference CML.event`.

Результатом применения функции является CML-событие, синхронизация по которому дает очередной вывод, построенный стратегией. В следующих двух пунктах будут описаны типы данных и функции, необходимые для комбинирования стратегий и написания атомарных стратегий.

2.3.6. Атомарные стратегии

Для программирования атомарных стратегий используется функция

`atomStrat:clauseFramework->(environment->unit)->strategy`.

Первый аргумент определяет интерфейс построенной стратегии. Второй аргумент — функцию типа `environment -> unit` — назовем телом атомарной стратегии. При вызове функции-тела производятся все те действия, которые должна выполнять атомарная стратегия, т. е. взаимодействие с окружением и всю внутреннюю обработку, предусмотренную семантикой. Для вычисления тела атомарной стратегии при ее инициализации порождается отдельный легковесный процесс (thread). Перечислим средства, предусмотренные для программирования тел атомарных стратегий. Тип

`environment`

служит для представления окружений, в которых функционируют атомарные стратегии. Внутри тела атомарной стратегии можно определить интерфейс, с которым она вызвана:

`myInterface: environment -> clauseFramework`.

Для представления мультимножеств формул, получаемых атомарной стратегией от окружения, используются элементы типа

`inputColl`.

Для индексирования таких мультимножеств используются элементы типа

`index`.

Функция

`inputEvt: environment -> string -> inputColl CML.event`,

беря в качестве аргументов окружение и идентификатор переменной со входной ориентацией, дает в качестве результата CML-событие, синхронизация по которому позволяет считывать очередное мультимножество формул по соответствующему входному каналу. Если второй аргумент не является идентификатором, входящим в интерфейс атомарной стратегии со входной ориентацией, то порождается исключение

`InvalidEntryId`.

Над элементами `inputColl` определены следующие операции:

```
entryId: inputColl -> string,
indices: inputColl -> index list,
retrieve: inputColl -> index -> form,
freeVars: inputColl -> freeVars.
```

Функция

`entryId`

позволяет определить идентификатор входного канала, по которому получено мультимножество. Функция

`indices`

возвращает список индексов всех элементов мультимножества. Извлечение формулы по ее индексу в мультимножестве производится функцией

`retrieve`.

Если индекс указан некорректно, т. е. мультимножество не содержит формулы с таким индексом, то порождается исключение

`WrongIndex`.

Функция

`freeVars`

позволяет найти множество свободных переменных, от которых зависят формулы, входящие в мультимножество. По этому поводу заметим, что в нашей системе каждое входное мультимножество является подмножеством некоторого выходного места некоторого клоза. Считаем, что формула из выходного места клоза зависит от всех свободных переменных, входящих в клоз, т. е., возможно, и от переменных, не входящих в нее. Поэтому `freeVars` вычисляет множество переменных, входящих в тот клоз, выходное место которого содержит соответствующее входное мультимножество.

Для представления содержимого выходных мест используется тип

`outputColl`.

Одноименная функция

```
outputColl:environment -> string -> (form list) -> outputColl
```

позволяет связать список формул с некоторым выходным местом. Если второй аргумент не является идентификатором, входящим в интерфейс атомарной стратегии с выходной ориентацией, то порождается исключение

`InvalidOutcomeId`.

Построение атомарной стратегией вывода и его возврат окружению производится с помощью функции `return`:

```
environment ->
  ((inputColl *(index list)* subst)list)*(outputColl list)->
    unit CML.event.
```

Вторым аргументом этой функции является пара списков, определяющих содержимое входных и выходных мест корневого клоза результирующего вывода. Тройки типа

```
inputColl * (index list) * subst
```

соответствуют парам

$$< \Phi, \theta > \in Mult^{+}(Form) \times Subst$$

в формальной семантике с тем лишь отличием, что они несут дополнительную информацию о том, в каком входном месте они могут содержаться.

2.3.7. Комбинирование стратегий

Для создания элементов типа

```
preemptDir,
```

представляющих директивы предпочтения, используются функции

```
somePD: string -> preemptDir,
allPD: string -> preemptDir,
orPD: preemptDir * preemptDir -> preemptDir,
orelsePD: preemptDir * preemptDir -> preemptDir.
```

Операции языка описания стратегий — пометка, перестановка, переименование, параллельная композиция и резолютивное замыкание — реализованы в виде функций

```
setLabel: string -> strategy -> strategy,
perm: clauseFramework * strategy -> strategy,
rename: formArg * strategy -> strategy,
parComp: strategy list -> strategy,
resolve: preemptDir * strategy -> strategy.
```

Если в вызове `perm` схема клоза не совместна с интерфейсом стратегии, то порождается исключение

```
ClauseFrameworkDoesNotMatchStrategyInterface.
```

Если в вызове `rename` формальный аргумент не соответствует интерфейсу стратегии, то порождается исключение

`FormArgDoesNotMatchStrategyInterface`.

Если в вызове `parComp` интерфейсы стратегий не совместны, то порождается исключение

`InconsistentInterfaces`.

Если в вызове `resolve` директива предпочтения содержит идентификаторы, не входящие в интерфейс стратегии одновременно со входной и выходной ориентацией, то порождается исключение

`PreemptDirDoesNotMatchStrategyInterface`.

Функция

```
setInfo: stratInfo -> strategy -> strategy
```

позволяет пометить стратегию значением типа `stratInfo`.

Кроме основных операций языка описания стратегий в реализацию системы из соображений практичности введены некоторые дополнительные возможности. Функции

```
bufferOut: int -> strategy -> strategy
```

и

```
bufferIn: (string * int) list -> strategy -> strategy
```

предназначены для буферизации выходных и входных каналов. Первый аргумент функции `bufferOut` определяет размер буфера выходного канала. Если он отрицателен, то буфер неограниченный. Первым аргументом `bufferIn` является список пар вида (v, n), где n определяет размер буфера входного канала с идентификатором v. Если указан отрицательный размер буфера, то буфер неограниченный.

Фильтрование входных и выходных данных осуществляется с помощью функций

```
filterIn: (string * (form list -> bool))list ->
```

```
strategy -> strategy
```

и

```
filterOut: (inference -> bool) -> strategy -> strategy.
```

Если p — некоторый предикат на выводах, s — стратегия, то стратегия `filterOut` p s будет возвращать те из выводов, порождаемых стратегией s, для которых выполняется условие p.

Пусть s — некоторая стратегия, l — список $[(v_1, p_1), ..., (v_n, p_n)]$ типа `(string * (form list -> bool))list`, причем все v_i попарно различны. Стратегия $s' =$ `filterOut` l s имеет тот же интерфейс, что и s. При инициализации s' инициализируется s; s' возвращает те же выводы, что и s. Если идентификатор $v \in In(cf)$ не является одним из v_i, то входной канал стратегии s, помеченный v, совпадает с соответствующим каналом стратегии s'. При попытке чтения стратегией s через канал, помеченный v_i, производится, возможно многократное, чтение стратегией s' через свой канал, помеченный тем же идентификатором. Попытки чтения производятся до тех пор, пока не будет считан список формул, удовлетворяющий p_i, после чего эти данные передаются стратегии s. Если список l содержит повторяющиеся идентификаторы, то вызов `filterIn` l порождает исключение

```
DuplicateIdOccurence.
```

ЗАКЛЮЧЕНИЕ

Идея управления поиском вывода для построения практичных систем поддержки доказательств не нова. Уже на заре автоматического доказательства теорем исследователи и экспериментаторы в этой области осознали, что использование только универсальных чисто автоматических методов не может помочь в решении большинства актуальных прикладных задач. Это дало толчок развитию направления, связанного с разработкой систем, в которых пользователь интерактивно строит доказательства, перепоручая программе выполнение лишь достаточно простых, но рутинных операций. Такие системы стали называть интерактивными доказывателями (в англоязычной литературе наряду с interactive theorem prover иногда используется термин proof-checker). Одной из первых таких систем была известная система Бойера и Мура (см. [1]). Несмотря на то, что рутинная часть работы при

построении доказательств в этой системе выполняется автоматически с помощью резолюции, некоторых достаточно универсальных методов обращения с равенством и эвристик для структурной индукции, при построении более или менее сложных доказательств невозможно обойтись без управления поиском вывода посредством указания большого количества промежуточных утверждений, что фактически является интерактивным построением доказательств. Наша система, как видно из примера 4, допускает такой стиль планирования доказательств. Более того, в нашей системе не обязательно указывать некоторые леммы явно, часто достаточно только указать, каким способом эти леммы доказываются и применяются. Это, с одной стороны, может иногда избавить от необходимости тщательно выписывать леммы и, с другой стороны, помогает в тех случаях, когда явный вид леммы трудно найти. В 1979 г. появился интерактивный доказыватель LCF (см. [2]), основанный на поиске доказательства сверху вниз (от цели к подцелям) с помощью так называемых тактик. Тактиками в LCF являются процедуры, соответствующие применению правил логики первого порядка (в том числе и производных) снизу вверх. Базовые тактики для основных правил предопределены в системе, тактики для производных правил можно программировать, используя базовые тактики, специальные операции, называемые тактикалами, и конструкции языка ML, который является родоначальником целого семейства языков программирования: Standard ML, Haskel, CAML и др. (в более поздней версии LCF — Cambridge LCF — используется уже Standard ML).Таким образом, LCF является "программируемым" доказывателем. Процесс построения доказательства в LCF выглядит следующим образом: пользователь указывает целевое утверждение и затем применяет к нему тактики, порождая подцели, строя при этом некоторое дерево. Процесс продолжается до тех пор, пока все листья дерева не станут аксиомами. Такой стиль поиска доказательств, называемый поиском, основанным на тактиках (tactical theorem proving), хорошо зарекомендовал себя в применениях, связанных с использованием логик, которые трудно автоматизировать (например логики высших порядков), и является основой целого клона современных систем поддержки доказательств: LCF, Isabelle, HOL и др. Несмотря на многие очевидные достоинства, такой стиль построения доказательств имеет свои недостатки. Один из них — жесткая ориентация на построение вывода в одном направлении — сверху вниз, т. е. от цели к подцелям. Другая сложность связана с необходимостью со стороны

пользователя осуществлять навигацию по дереву вывода, которое обычно быстро разрастается. При этом ядро решаемой проблемы часто теряется во множестве несущественных деталей. В этом смысле наш подход, связанный с направлением автоматического поиска вывода посредством описания стратегий, представляется более подходящим для реализации логик, для которых имеются хорошие дедуктивные процедуры. Кроме того, наши стратегии-процессы в отличие от тактик-функций в стиле LCF могут накапливать данные, необходимые для решения задачи (см. стратегию Unify из примера 5), что позволяет естественным и эффективным способом описывать некоторые дедуктивные процедуры. Для реализации различных логик во многих известных системах используется подход, связанный с описанием реализуемой (объектной) логики в рамках базовой логики системы (металогики). В качестве металогики обычно используется вариант логики высших порядков или теории типов. Объектная логика описывается аксиомами в языке металогики. Доказательствам в объектной логике сопоставляются доказательства в металогике. Очевидным достоинством такого подхода к реализации объектных логик в сравнении с нашим подходом является декларативность. Кроме того, доказательство в металогике обычно полностью формализовано и не требует дополнительной проверки. Но у данного подхода есть и определенные издержки, в частности, поиск доказательств в металогике обычно довольно трудно автоматизировать. Требование построения детального формального доказательства задтрудняет реализацию проблемно-ориентированных дедуктивных процедур и их интеграцию в рамках единого доказывателя. Кроме того, усложняются синтаксические объекты, которыми должен манипулировать пользователь, и, как следствие, теряется наглядность.

СПИСОК ЛИТЕРАТУРЫ

1. **Boyer R.S., Moore J.S.** A Computational Logic. — New York: Academic Press, 1979.
2. **Paulson L.C.** Logic and Computation: Interactive Proof with Cambridge LCF // Cambridge Tracts in Theoretical Computer Science 2. — Cambridge University Press, 1987.
3. **Paulson L.C.** The foundation of a generic theorem prover. — Univ. Cambridge, 1988. — (Computer Laboratory. Tech. Rep. N 130).
4. **Paulson L.C.** A preliminary user's manual for Isabell. — Univ. Cambridge, 1988. — (Computer Laboratory. Tech. Rep. N 133).
5. **Crow J., Owre S., Rushby J., Shankar N., Srivas M.** A Tutorial Introduction to PVS // Proc. of the Workshop on Industrial-Strengh Formal Specification

Techniques. — Baco Raton, Florida, 1995.
6. **Harper R.** Introduction to Standard ML. — Univ. Edinburgh, Laboratory for Foundation of Computer Science, revised edition, 1989.
7. **Reppy J.H.** Higher-Order Concurrency. — Cornell Univ.,1992. — (Computer Sci. Department. Tech. Rep. N TR92-1285).
8. **Непомнящий В.А., Рякин О.М.** Прикладные методы верификации программ. — М.: Радио и связь, 1988.
9. **Nelson G., Oppen D.C.** Simplification by Cooperating Decision Procedures // ACM Trans. on Programming Languages and Systems. — 1979. — Vol. 1, N 2.

А. Е. Рязанов

СИСТЕМА БУЦЕФАЛ: КОМБИНИРОВАНИЕ ДЕДУКТИВНЫХ ПРОЦЕДУР И ОПИСАНИЕ СТРАТЕГИЙ ПОИСКА ДОКАЗАТЕЛЬСТВ

Препринт
50

Рукопись поступила в редакцию 03.03.98
Рецензент В. А. Непомнящий
Редактор Л. А. Карева

Подписано в печать 10.08.98
Формат бумаги 60×84 1/16 Объем 2,1 уч.-изд.л., 2,3 п.л.
Тираж 75 экз.

Отпечатано на ризографе "AL Group", 630090, пр. Акад. Лаврентьева, 6