Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математической кибернетики

Ларочкин Петр Викторович

Решение задачи проверки модели в исчислении индуктивных конструкций

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
к.ф.-м.н., доцент
Подымов В. В.

Москва, 2023

Содержание

1. Введение **4**

2. Основные понятия **6**

2.1. Логика ветвящегося времени (CTL) 6

2.2. Исчисление индуктивных конструкций и доказательство теорем 8

2.2.1. Исчисление конструкций 8

2.2.2. Индуктивные типы 10

2.2.3. Сопоставление по шаблону 11

2.2.4. Рекурсивное отображение 12

2.2.5. Исчисление индуктивных конструкций (CIC) 13

2.2.6. Соответствие Карри–Говарда 13

3. Постановка задачи **15**

4. Основная часть **15**

4.1. Постановка задачи проверки модели в исчислении индуктивных конструкций 15

4.1.1. Вспомогательные индуктивные типы 15

4.1.2. Основные индуктивные типы 18

4.1.3. Постановка задачи проверки модели 23

4.2. Синтаксис и семантика языка тактик *Ltac*. 25

4.3. Синтаксис и семантика базовых тактик 29

4.4. Ограничения на представление модели Крипке 34

4.5. Алгоритмы (тактики) 36

4.5.1. solve_fV 36

4.5.2. solve_fOr 37

4.5.3. solve_fAnd 38

4.5.4. solve_fAX 39

4.5.5. solve_fAU 41

4.5.6. Главная тактика applicator 49

4.5.7. Главная тактика 51

4.6. Корректность главной тактики 52

4.7. Полнота главной тактики 52

5. Полученные результаты **57**

Список литературы **58**

1. Введение

Темпоральная логика — логика, язык которой содержит средства описания взаимосвязей логических значений, изменяющихся с течением времени [1]. Распространенными темпоральными логиками являются логика линейного времени (*LTL*) и логика ветвящегося времени (*CTL*). Их обобщением является расширенная логика ветвящегося времени (*CTL**). Темпоральные логики активно применяются в задачах верификации программ (проверки свойств программ). Утверждения о свойствах программы можно описать с помощью языка темпоральной логики. Программу можно описать как систему с конечным числом состояний и отношением перехода. Одним из вариантов такой системы может быть модель Крипке. Задача проверки выполнимости построенных таким образом утверждений на модели Крипке называется проверка модели (*model checking*).

Coq – интерактивное средство доказательства теорем. Это средство позволяет строить и доказывать теоремы, и оно же проверяет правильность построенного доказательства, обеспечивая этим корректность доказанной теоремы с высокой степенью доверия - такой же, как и к правильности работы самого средства *Coq*. Правильность работы *Coq* обеспечивается тем, что ядро *Coq* реализовано довольно простым способом, чтобы сократить вероятность ошибки, к тому же теория, лежащая в основе, является надежной. *Coq* активно применяется для верификации программного обеспечения [2] [3] и для доказательств утверждений из математики[4].

Основным инструментом для построения доказательств в *Coq* являются тактики. Тактики – это особые команды для автоматизированного построения доказательства теоремы, причем такие, что могут быть реализованы и самим пользователем. В *Coq* одним из инструментов, позволяющих описывать новые тактики, является встроенный язык тактик *Ltac*[5].

Существует работы [6][8][9], посвященные анализу темпоральных логик в *Coq*. В работе [9] автор реализует операторы *LTL* и доказывает связанные с ними свойства. В работе [6] автором была предложена формализация модели Крипке и формул *CTL** в *Coq*. При помощи этой формализации доказаны средствами *Coq* некоторые свойства языка *CTL** и соответствующего варианта задачи проверки модели. В работах [8][7] автор провел обширное исследование, в котором удачно формализовал операторы *CTL*, модель

Крипке и реализовал некоторые логические системы в *Coq*. Главным результатом этой работы является утверждения о корректности и полноте некоторых систем доказательств относительно задачи проверки выполнимости формул *CTL* на моделях Крипке. Стоит отметить, что в этой работе построены новые тактики для применения в доказательствах утверждений о конечных множествах. Однако ни одна из приведенных работ не предоставляет автоматического средства доказательства утверждения о выполнимости формулы на модели Крипке. Такое средство (тактика) представляет интерес по крайней мере по двум причинам. Во-первых, в качестве вспомогательного инструмента для доказательства теорем, если в процессе доказательства возникает необходимость обосновать выполнимость конкретной формулы на конкретной модели - аналогично широко применяющейся тактике *lia* [10], автоматически доказывающей существование решения конкретной системы линейных уравнений над целыми числами. Во-вторых, для исследования возможностей преобразования известных алгоритмов проверки модели в алгоритмы доказательства соответствующих теорем.

Целью магистерской диссертации была разработка и исследование средства автоматического доказательства теорем о выполнимости формул *CTL* на моделях Крипке. Для достижения этой цели требовалось решить следующие задачи:

1. Поставить обозначенную задачу выполнимости формально в терминах *Coq*.

2. Разработать тактику доказательства теоремы, обозначающей выполнимость заданной формулы на заданной модели в формальной постановке, на языке *Ltac* - то есть разработать алгоритм проверки модели в особой вычислительной модели, предназначенной для вывода логических суждений.

3. Описать формальные синтаксис и семантику фрагмента языка *Ltac*, потребовавшегося в разработанной тактике, основываясь на технической документации [11] и работе [5], содержащих описания, близкие к формальным, но не вполне строгие. Следует отметить, что формальные синтаксис и семантика *Ltac* изложены в диссертации [19], но это изложение настолько объёмно (102 страницы) и содержит столько деталей и возможностей языка, что его изучение и использование оказалось практически невозможным.

4. Обосновать корректность и исследовать полноту разработанной тактики.

Корректность означает, что если тактикой строится доказательство теоремы о выполнимости формулы на модели из формальной постановки задачи, то формула действительно выполняется на модели. Под исследованием полноты понимается строго обоснованное обозначение класса формул и моделей, для которых тактика успешно строит доказательство.

В разделе 4.1 опысывается постановка задачи проверки модели в терминах *Coq*. В разделах 4.2, 4.3 приводится описание части синтакстиса и семантики языка *Ltac*, а также необходимых базовых тактик. В разделе 4.4 описываются какие модели Крипке будут рассматриваться для решения задачи проверки модели. В разделе 4.5 приводится поэтапное описание тактики, доказывающей выполнимость формулы на модели. В разделе 4.6 и 4.7 приводится соответственно обоснование корректности и полноты построенной тактики для рассматриваемых моделей и формул определенного вида.

2. Основные понятия

2.1. Логика ветвящегося времени (CTL)

Понятия, изложенные в данном разделе, приводятся, например, в [1].

Пусть задано множество атомарных высказываний AP. Структура Крипке — это четверка (S, S_0, T, L), где

- S — конечное множество состояний
- S_0 — множество начальных состояний, $S_0 \subset S$
- T — $T \subseteq S \times S$ — отношение переходов, обладающее свойством тотальности: для любого состояния s_1 существует состояние s_2, такое что $(s_1, s_2) \in T$
- L — функция разметки состояния, $L : S \to 2^{AP}$.

Путь в модели Крипке M из состояния s_0 — это бесконечная последовательность состояний $\pi = s_0, s_1...$, где $(s_i, s_{i+1}) \in T$ $(i = 0, 1, 2...)$ модели Крипке M. Обозначим i-ый суффикс пути $\pi^i = s_i, s_{i+1}, ...$

Синтаксис логики ветвящегося времени (CTL) определяется следующей БНФ:

$$\phi ::= p|\neg\phi|\phi \vee \phi|\phi \wedge \phi|\phi \Rightarrow \phi|EX\phi|AX\phi|AF\phi|EF\phi|AG\phi|EG\phi|\phi AU\phi|\phi EU\phi|\phi AR\phi,$$

где $p \in AP$, ϕ— формула логики ветвящегося времени.

Выполнимость формулы ϕ в состоянии s модели M: $M, s \vDash \phi$ определяется следующим образом:

1) $M, s \vDash p \Leftrightarrow p \in L(s)$

2) $M, s \vDash \neg\phi \Leftrightarrow M, s \nvDash \phi$

3) $M, s \vDash \phi_1 \vee \phi_2 \Leftrightarrow M, s \vDash \phi_1$ или $M, s \vDash \phi_2$

4) $M, s \vDash \phi_1 \wedge \phi_2 \Leftrightarrow M, s \vDash \phi_1$ и $M, s \vDash \phi_2$

5) $M, s \vDash \phi_1 \Rightarrow \ \phi_2 \Leftrightarrow M, s \nvDash \phi_1$ и $M, s \vDash \phi_2$

6) $M, s \vDash EX\phi \Leftrightarrow$ в M существует путь π из состояния s, что $M, \pi^1 \vDash \phi$

7) $M, s \vDash EF\phi \Leftrightarrow$ в M существует такой путь π из состояния s, такое $k \geq 0$, что $M, \pi^k \vDash \phi$

8) $M, s \vDash EG\phi \Leftrightarrow$ в M существует такой путь π из состояния s, что для любого $k \geq 0$ верно, что $M, \pi^k \vDash \phi$

9) $M, s \vDash \phi_1 EU \phi_2 \Leftrightarrow$ в M существует путь π из состояния s, существует такое $k \geq 0$, что M, $\pi^k \vDash \phi_2$ и для каждого $0 \leq j < k$ верно соотношение $M, \pi^j \vDash \phi_1$

10) $M, s \vDash AX\phi \Leftrightarrow$ для любого пути π из состояния s в модели M верно соотношение $M, \pi^1 \vDash \phi$

11) $M, s \vDash AF\phi \Leftrightarrow$ для любого пути π из состояния s в модели M существует такое $k \geq 0$ верно, что $M, \pi^k \vDash \phi$

12) $M, s \vDash AG\phi \Leftrightarrow$ для любого пути π из состояния s в модели M, для любого $k \geq 0$ верно, что $M, \pi^k \vDash \phi$

13) $M, s \vDash \phi_1 AU \phi_2 \Leftrightarrow$ для любого пути π из состояния s в модели M верно, что существует такое $k \geq 0$, что M, $\pi^k \vDash \phi_2$ и для каждого $0 \leq j < k$ верно соотношение $M, \pi^j \vDash \phi_1$

14) $M, s \vDash \phi_1 AR \phi_2 \Leftrightarrow$ для любого пути π из состояния s в модели M верно, что для любого $k \geq 0$ верно соотношение $M, \pi^k \vDash \phi_2$ как минимум до тех пор, пока не будет верным соотношение $M, \pi^t \vDash \phi_1$, где $t \geq k$

15) $M, s \vDash \phi_1 ER \phi_2 \Leftrightarrow$ существует путь π из состояния s в модели M верно, что для любого $k \geq 0$ верно соотношение $M, \pi^k \vDash \phi_2$ как минимум до тех пор, пока не будет верным соотношение $M, \pi^t \vDash \phi_1$, где $t \geq k$

Далее, будет рассматриваться укороченный синтаксис CTL, состоящий из элементов $\{p,\ \neg,\ \rightarrow,\ AX,\ AR,\ AU\}$. Любая формула CTL может быть выражена с помощью этого набора, так как $\{\neg,\ \rightarrow\}$ – это полная система в алгебре логики, $\{AX,\ AU,\ EU\}$ – темпоральные операторы, через которые можно выразить остальные темпоральные операторы, как показано в [12] с учётом эквивалентности $(\phi_1 EU \phi_2) = \neg(\neg\phi_1 AR \neg\phi_2)$ можно заменить оператор EU на AR.

Пусть заданы модель Крипке $M = (S, S_0, R, L)$ и CTL-формула ϕ. Задача проверки модели заключается в том, чтобы проверить включение $S_0 \subset S_\phi$, где $S_\phi = \{s \in S | M, s \vDash \phi\}$.

2.2. Исчисление индуктивных конструкций и доказательство теорем

Понятия, изложенные в данном разделе, приводятся, например, в [13][14][15][16].

2.2.1. Исчисление конструкций

Основные объекты в исчислении конструкций – это термы. Синтаксис термов над множеством переменных X и множеством типов

$$Sorts = \{Prop\} \cup (\cup_{i \in N} \{Type_i\})$$

задается следующей БНФ:

$$term ::= \ x \mid s \mid (\lambda\ (x : term).\ term) \mid (term\ term) \mid \Pi(x : term),\ term,$$

где $s \in Sorts$, $x \in V$.

Каждому терму, построенному согласно описанной выше БНФ, ставится в соответствие тип (или, по-другому, говорится, что этот терм имеет тип), который также является термом. Запись «$y : K$» обозначает, что терм обозначенный через переменную y имеет тип K.

Контекстом будем называть конечное множество, элементы которого имеют вид $x : T$ или $x := t : T$, где x - переменная, t - терм и T - тип этого терма.

Свободная переменная – это переменная, которая не связана ни одним из кванторов λ, Π. Через $FV(M)$ обозначим множество свободных переменных в терме M.

Запись $t\{x/u\}$ обозначает терм t, в котором свободная переменная x заменена на терм u. Будем говорить, что терм $t\{x/u\}$ является уточнением терма t. Суждение о типе – это запись

$$\Gamma \vdash t : A,$$

где Γ – это контекст, t – это терм, A – это тип.

Основные правила вывода исчисления конструкций:

$$\frac{\Gamma \vdash}{\Gamma \vdash Prop : Type_1}, \quad \frac{\Gamma \vdash}{\Gamma \vdash Type_i : Type_{i+1}}, \quad \frac{\Gamma \vdash \quad x : A \in \Gamma}{\Gamma \vdash x : A}, \quad \frac{\Gamma \vdash A : s \quad x \notin \Gamma \quad s \in Sorts}{\Gamma, x : A \vdash},$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi(x : A), B}, \quad \frac{\Gamma \vdash f : \Pi(x : A), B \quad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B\{x/a\}}, \quad \frac{\Gamma, x : A \vdash B : Prop}{\Gamma \vdash \Pi(x : A), B : Prop},$$

$$\frac{\Gamma, x : A \vdash B : Type_i \quad \Gamma, \vdash A : Type_i}{\Gamma \vdash \Pi(x : A), B : Type_i}, \quad \frac{\Gamma \vdash t : T \quad x \notin \Gamma}{\Gamma, x := t : T \vdash},$$

где Γ– это контекст, $x \in V$, $Sorts = \{Prop\} \cup (\cup_{i \in N}\{Type_i\})$.

Запись $\Gamma \vdash$ является суждением о типе, которое означает, что контекст Γ построен корректно (согласно правилам). Суждение такого вида будем называть ослабленным суждением о типе. Будем говорить, что терм t типа T выводим в контексте Γ, если можно построить последовательность суждений о типе, построенные с помощью правил, согласно которым верно следующее суждение о типе

$$\Gamma \vdash t : T.$$

Согласно устройству основных правил, каждому терму t можно сопоставить реализуемое им отображение или множество реализуемых объектов.

- Терму $(\lambda\ (x\ :\ term_1).\ term_2)$ отвечает отображение с аргументом x типа $term_1$, возвращающее для значения v этого аргумента значение выражения $term_2$ на v.
- Терму $(term_1\ term_2)$ отвечает результат применения отображения, записанного как $term_1$, к аргументу, записанному как $term_2$.

- Терму $\Pi(x : term_1),\ term_2$ отвечает семейство типов (термов), которые образуются при подстановке типа x в $term_2$.
- Запись «$A \to B$» обозначает терм $\Pi(x : A).B$, где $x \notin FV(B)$.
- Запись «x», $x \in V$, означает переменную x, имеющую тип, заданный контекстом.
- Запись «s», $s \in Sorts$, обозначает тип из заданного множества $Sorts$, тип которого определяется согласно правилам вывода.

Будем использовать следующее двуместное отношение β-редукции $\xrightarrow{\beta}$ на множестве термов, которое определяется следующим образом:

$$(\lambda\ (x : A).\ M)\ N \xrightarrow{\beta} M\{x/N\}$$

2.2.2. Индуктивные типы

Пусть $Names$ и $Constr$ – это соответственно множество имен типов и имен конструкторов. Индуктивный тип – это запись

$$((I\ parameters :\ Ar),\ constructors),$$

где

- $I,\ I \in Names$ – имя индуктивного типа,
- $parameters = p_1 : P_1,\ ...,\ p_n : P_n$ – это параметры индуктивного типа I,
- $Ar = \Pi(q_1 : Q_1), ..., \Pi(q_w : Q_w), s$ – это тип этого индуктивного типа, $s \in Sorts$,
- $constructors = \{constr_1 : c_1, ..., constr_l : c_l\}$ – это конечное множество конструкторов, которые имеют имя $constr_i$, $constr_i \in Constr$, и тип c_i, который имеет следующий вид (разные для каждого конструктора)

$$c_i = \Pi\ parameters, (\Pi(x_1 : A_1), (\Pi(x_2 : A_1), ...(\Pi(x_k : A_k),\ I\ parameters)...)).$$

Обозначим через Γ_I контекст

$$\Gamma_I = [(I\ parameters :\ Ar),\ (constr_1 : c_1), ..., (constr_m : c_m)].$$

Для каждого используемого индуктивного типа I и каждого конструктора $constr_i$ описанного выше вида в исчисление индуктивных конструкций добавляется правило

$$\frac{}{\Gamma_I \vdash constr_i\ x_1\ ...\ x_k : I\ parameters}.$$

То есть запись $(constr_i\ x_1\ ...\ x_k)$ для конструктора и для аргументов соответствующих типов является термом этого типа. Множество термов этого индуктивного типа состоит из всех термов вида $(constr_i\ t_1\ ...\ t_k)$, для которых можно вывести суждение о том, что эти термы имеют тип $I\ parameters$ в контексте Γ_I. Указанный терм $constr_i\ t_1\ ...\ t_k$ будем называть термом, построенным конструктором $constr_i$.

Индуктивный тип также имеет дополнительные ограничения корректности построения [13], однако для дальнейшего описания эти ограничения не потребуются. Далее будут использоваться индуктивные типы, которые выполняют эти ограничения и проверены средством *Coq*.

Будем записывать $x \in y$, если терм x является подтермом терма y. Например, рассмотрим индуктивный тип Nat натуральных чисел, который можно описать следующим образом:

$$(Nat : Type, \{O : Nat, S : Nat \to Nat\}).$$

Контекст

$$\Gamma_{Nat} = [Nat : Type, O : Nat, S : Nat \to Nat]$$

Согласно, правилам вывода можно построить следующие суждения о типе

$$\Gamma_{Nat} \vdash O : Nat$$

$$\Gamma_{Nat} \vdash (S\ O) : Nat$$

$$\Gamma_{Nat} \vdash (S\ (S\ O)) : Nat$$

...

Таким образом, термами индуктивного типа Nat являются $O,\ (S\ O),\ (S\ (S\ O))$...

2.2.3. Сопоставление по шаблону

Пусть:

- $I\ parameters$ – индуктивный тип I с параметрами $parameters$
- $f_1, ..., f_n$ – термы типа P, $FV(P) = \emptyset$
- для каждого i, множество $FV(f_i) = \{u_{i_0}, u_{i_1}, ..., u_{i_m}\}$
- $x\ :\ I\ parameters$ – переменная x типа $I\ parameters$
- c_i – i-ый конструктор индуктивного типа $I\ parameters$, с аргументами

$$x_{i_1}\ :\ A_{i_1}, ..., x_{i_m}\ :\ A_{i_m}$$

Тогда будем считать термом следующую запись

$$\begin{aligned}
&match\ x : (I\ parameters)\ with\\
&|\ c_1\ x_{1_1}...x_{1_m} \Rightarrow f_1\{u_{1_0}/x\}\{u_{1_1}/x_{1_1}\}...\{u_{1_m}/x_{1_m}\}\\
&...\\
&|\ c_n\ x_{n_1}...x_{n_m} \Rightarrow f_n\{u_{n_0}/x\}\{u_{n_1}/x_{n_1}\}..\{u_{n_m}/x_{n_m}\}\\
&end : P
\end{aligned}$$

Указанный выше терм (обозначим его через m) имеет тип $(I\ parameters) \to P$. Терму m отвечает отображение, принимающее аргумент x типа $(I\ parameters)$ и возвращающее значение v типа P, устроенное следующим образом: если $x = c_i\ x_{i_1},\ ...,\ x_{i_m}$, то $v = f_i\{u_{i_0}/x\}\{u_{i_1}/x_{i_1}\}...\{u_{i_m}/x_{i_m}\}$.

Будем использовать двуместное отношение ι-редукции $\xrightarrow{\iota}$ на множестве термов, которое определяется следующим образом:

$$((m)\ t) : P\ \xrightarrow{\iota}\ f_k\{u_{k_0}/t\}\{u_{k_1}/x_{k_1}\}...\{u_{k_m}/x_{k_m}\},$$

где t построен k-ым конструктором индуктивного типа $I\ parameters$.

2.2.4. Рекурсивное отображение

Пусть:

- $\{A_1, ..., A_{n-1}, A_{n+1}, ..., A_m\}$– типы, A_n – индуктивный тип,

- терм t имеет тип B в контексте $[(f : \Pi(x_1 : A_1), ..., \Pi(x_m : A_m), B), (x_1 : A_1)...(x_m : A_m)]$,
- в записи $f\ u_1\ ...\ u_m$ в терме t, терм u_n структурно меньше чем x_n, то есть $u_n \in x_n$ и $x_n \neq u_n$

Тогда будем считать термом следующую запись:

$$fixpoint\ f\ (x_1 : A_1)...(x_m : A_m)\{struct\ x_n\} : B := t$$

Этот терм имеет тип $(\Pi(x_1 : A_1), ..., \Pi(x_m : A_m), B)$, и ему отвечает отображение, для аргументов $u_1...u_m$ типов $A_1...A_m$ соответственно возвращающее значение v, устроенное следующим образом:

1) Рассматривается терм t, в котором после подстановки переменных $x_1, ..., x_m$ записи вида $f\ u_1...u_m$ заменяются на термы вида $t\{u_1/x_1\}...\{u_m/x_m\}$.
2) К этому терму применяются правила β и ι редукции. Процесс повторяется до тех пор, пока к терму нельзя будет применить правила редукции и пока в терме присутствуют записи вида $f\ u_1...u_m$.
3) Искомое значение v - это последний полученный терм, к которому невозможно применить правила редукции и в котором нет записи вида $f\ u_1...u_m$.

Построенный терм может быть включен в контекст и имеет следующий вид:

$$f := (fixpoint\ f\ (x_1 : A_1)...(x_m : A_m)\{struct\ x_n\} : B := t) : (\Pi(x_1 : A_1), ..., \Pi(x_m : A_m), B).$$

2.2.5. Исчисление индуктивных конструкций (CIC)

Исчисление индуктивных конструкций (CIC) – это расширение исчисления конструкций, в котором есть возможность определять индуктивный тип, строить термы, используя дополнительно выражения $match$ и $fixpoint$.

2.2.6. Соответствие Карри–Говарда

Соответствие Карри-Говарда применительно к исчислению индуктивных конструкций - это соответствие между элементами исчисления индуктивных конструкций и исчислений,

предназначенных для конструктивного доказательства высказываний на языке логики предикатов. Это соответствие устроено следующим образом.

- Квантору П сопоставляется квантор всеобщности.
- Каждый тип A типа $Prop$ расценивается как формула логики предикатов той же структуры. Например, тип $A \to B$ расценивается как формула логики предикатов $A \Rightarrow B$, где A и B имеют тип $Prop$.
- Запись $(t : A)$, где t - терм и A - тип, трактуется как утверждение о том, что t является доказательством утверждения A.
- Запись $(x := t : A)$ трактуется как присвоение имени x доказательству t утверждения A.
- Суждение о типе $\Gamma \vdash t : A$ трактуется как утверждение о том, что t является доказательством утверждения A в предположении о наличии доказательств, записанных в контексте Γ.

  Например, пусть задан контекст

  $$\Gamma = \Gamma_{Nat} \cup [A : Prop, B : Prop, (l : Nat \to B), (p : B \to A)],$$

  где A, B – утверждения, l – доказательство утверждения, того что для любого терма типа Nat верно утверждение B, p – доказательство утверждения $B \Rightarrow A$. Выведем терм доказательства для суждения A, используя правила вывода. То есть покажем суждение $\Gamma \vdash t : A$ выводимо.

  $$\dfrac{\dfrac{\Gamma \vdash O : Nat}{\Gamma \vdash (l\ O) : B}}{\Gamma \vdash (p\ (l\ O)) : A}$$

  Таким образом, выводимо суждение $\Gamma \vdash t : A$, где $t = (p\ (l\ O))$. То есть t является доказательством утверждения A в предположении о наличии доказательств, записанных в контексте Γ.

3. Постановка задачи

В магистерской диссертации требовалось выполнить следующее:

1. Поставить формально задачу проверки модели, изложенную в разделе 2.1, в терминах исчисления индуктивных конструкций.

2. Предложить формальные синтаксис и семантику фрагмента языка *Ltac* и базовых тактик, которые потребуется использовать для решения задачи проверки модели в терминах исчисления индуктивных конструкций.

3. Разработать алгоритм, записанный в терминах формализованных элементов языка *Ltac*, для доказательства выполнимости или невыполнимости любой заданной формулы на заданной модели Крипке (или для формул и моделей какого-либо достаточно нетривиального подкласса).

4. Обосновать корректность и полноту разработанного алгоритма.

4. Основная часть

4.1. Постановка задачи проверки модели в исчислении индуктивных конструкций

Данный раздел посвящен постановке задачи проверки модели относительно формул *CTL* в терминах исчисления индуктивных конструкций.

4.1.1. Вспомогательные индуктивные типы

В этом подразделе приводится математическое описание индуктивных типов, лежащие в основе программного кода из стандартной библиотеки *Coq*[17].

Формализация квантора существования в терминах *CIC*.

Индуктивный тип *ex*, который соответствует квантору существования логики

предикатов, устроен так:

$$((ex(A : Type)(P : A \to Prop) : Prop), \{$$
$$(ex_{intro} : \Pi(A : Type), \Pi(P : A \to Prop), \Pi(x : A), P\ x \to ex\ A\ P)\}).$$

Построение терма типа ex возможно с помощью единственного конструктора ex_{intro} с параметрами:

1) тип терма A, для которого нужно показать существование,

2) формулировка P утверждения о терме,

3) терм типа A и доказательство суждения P для него.

Таким образом, для того, чтобы доказать что существует терм, удовлетворяющий утверждению P, необходимо предоставить сам терм и доказательство выполнения термом формулировки P для этого терма.

Индуктивный тип $ex2$, который также соответствует квантору существования логики предикатов, устроен так:

$$((ex2 : \Pi(A : Type), \Pi(P : A \to Prop), \Pi(Q : A \to Prop), Prop), \{$$
$$(ex_{intro2} : \Pi(A : Type), \Pi(P : A \to Prop), \Pi(Q : A \to Prop),$$
$$\Pi(x : A), P\ x \to Q\ x \to ex2\ A\ P\ Q\}))\}).$$

Построение терма типа $ex2$ схоже с построением терма типа ex, но еще требуется доказательство для формулировки Q.

Введем обозначение «*exists y: A, p*», которое означает индуктивный тип «$exists\ A\ (\lambda(y : A).p)$». Введем обозначение «*exists2 x : A, p & q*», которое означает индуктивный тип «$exists2\ A\ (\lambda(x : A).p)(\lambda(x : A).q)$».

Формализация дизъюнкции в терминах *CIC*

Индуктивный тип or, который соответствует дизъюнкции из алгебры логики, устроен так:

$$((or(A : Prop)(B : Prop) : Prop), \{$$
$$(or\_introl : \Pi(A : Prop), \Pi(B : Prop), A \to or\ A\ B),$$
$$(or\_intror : \Pi(A : Prop), \Pi(B : Prop), B \to or\ A\ B)\}).$$

Построение терма типа *or* возможно с помощью одного из его конструкторов, которым требуется доказательство одного из двух утверждений.

Введем обозначение «$A \vee B$» , которое означает тип *(or A B)*.

Формализация конъюнкции в терминах *CIC*

Индуктивный тип *and*, который соответствует конъюнкции из алгебры логики, устроен так:

$$((and(A : Prop)(B : Prop) : Prop), \{$$
$$(conj : \Pi(A : Prop), \Pi(B : Prop), A \to B \to and\ A\ B)\}).$$

Построение терма типа *and* возможно с помощью одного конструктора, которому требуется доказательство двух утверждений.

Введем обозначение «$A \wedge B$» , которое означает тип *(and A B)*.

Формализация отношения $=$ в терминах *CIC*

Индуктивный тип *eq*, который соответствует двуместному отношению $=$ для термов, устроен так:

$$((eq(A : Type)(x : A) : A \to Prop), \{$$
$$(eq_{refl} : \Pi(A : Type)(x : A), eqAxx),$$
$$\}).$$

Терм типа *eq A x y* можно построить только в том случае, если терм x типа A идентичен терму y.

Введем обозначение «$(x : A) = (y : A)$» , которое означает тип *(eq A x y)*.

Формализация отношения $<$ в терминах *CIC*

Индуктивный тип le, который соответствует двуместному отношению $\leq$ на множестве натуральных чисел, устроен так:

$$((le(n : nat) : nat \to Prop), \{$$
$$(le_n : \Pi(n : nat), le\ n\ n),$$
$$(le_S : \Pi(n : nat), \Pi(m : nat), le\ n\ m \to le\ n\ (S\ m))\}).$$

Терм типа le строится с помощью конструкторов следующим образом: если два числа являются равным, то применяется конструктор le_n, если два числа не являются равными, тогда для построения нужно рекурсивно построить доказательства конструктором le_S, того что отношение выполняется для второго аргумента на единицу меньше до тех пор, пока два терма не станут равными.

Введем обозначение «$n < m$» , которое означает тип *(le (S n) m)*.

Формализация $false$ в терминах *CIC*

Индуктивный тип $False$, который соответствует логическому значению $false$ из алгебры логики, устроен так:

$$((False : Prop), \{\}).$$

Терм индуктивного типа $False$ невозможно построить.

4.1.2. Основные индуктивные типы

В этом подразделе приводится математическое описание индуктивных типов из работы [8][7][18].

Формализация модели Крипке в терминах *CIC*

Индуктивный тип sts, соответствующий моделям Крипке, устроен так:

$$((sts : Type), \{$$
$$(STS :$$
$$\Pi(state : Type),$$
$$\Pi(trans : state \to state \to Prop),$$
$$(state \to Prop) \to$$
$$(nat \to state \to Prop) \to$$
$$(\Pi(w : state), exists(v : state), trans\ w\ v))\}).$$

Построить терм типа sts можно с помощью единственного конструктора STS. Аргументами конструктора являются:

1) Тип $state$, термы которого являются состояниями этой модели Крипке. Множество термов соответствует множеству S из раздела 2.1.

2) Отображение $trans$, которое является отношением переходов $trans$ соответствует отношению T из раздела 2.1.

3) Третий аргумент – это отображение $init$ из множества состояний в $Prop$. Этот аргумент был добавлен в конструктор для соответствия с моделью Крипке из раздела 2.1. И множество термов t, для которых можно доказать $init\ t$, соответствует множеству S_0 из раздела 2.1.

4) Четвертый аргумент – это отображение натурального числа, соответствующего элементу из множества атомарных высказываний, в предикат над множеством состояний. Соответствует функции разметки L из раздела 2.1.

5) Пятый аргумент – это доказательство тотальности построенной модели Крипке.

Отображение, возвращающее тип состояний модели Крипке M, отвечает терму

$$\lambda(M : sts).$$
$$match\ M\ with$$
$$|\ STS\ state\ trans\ init\ label\ serial \Rightarrow state$$
$$end$$

Обозначим это отображение как $state$. Аналогично устроены термы, отвечающие отображениям, возвращающим отношение переходов ($trans$), множество начальных состояний модели ($init$), функцию разметки состояний ($label$), доказательство тотальности отношения переходов ($serial$).

Формализация оператора AX в терминах *CIC*

Пусть контекстом задана модель Крипке типа sts, обозначенная переменной H. Введем обозначения:

- «$Transition$» обозначает $trans\ H$.
- «$State$» обозначает $state\ H$.

Терм (обозначим его через cAX_{term}), соответствующий оператору AX из CTL, который имеет тип (обозначим его тип через cAX_{type})

$$(State \to Prop) \to State \to Prop,$$

устроен так:

$$\lambda(p : State \to Prop).$$
$$\lambda(w : State).\ \Pi(v : State), Transition\ w\ v \to\ p\ v.$$

Если отображение $f : State \to Prop$ трактовать как множество состояний (всех термов st типа $State$, для которых верно $(f\ st) : Prop$), то, как можно видеть по структуре терма и устройству определения операции AX в разделе 2.1, возвращаемое отображение - это множество всех состояний модели, в которых выполняется формула $AX\phi$, если p - множество всех состояний, в которых выполняется ϕ.

Определим контекст

$$\Gamma_{AX} = [cAX := cAX_{term} : cAX_{type}].$$

Формализация пути в терминах *CIC*

Терм (обозначим его через $path_{term}$), соответствующий определению пути из секции 2.1, который имеет тип (обозначим его тип через $path_{type}$)

$$(nat \to State) \to Prop,$$

устроен так:

$$\lambda(pi : nat \to State).$$

$$\Pi n : Nat, Transition\ (pi\ n)(pi\ (S\ n)).$$

При подстановке отображения pi, означающего последовательность состояний, он возвращает тип. Терму этого типа отвечает отображение, которое ставит любому натуральному числу n терм типа $Transition\ (pi\ n)(pi\ (S\ n))$ для n-ого и $(n+1)$-ого состояния последовательности pi. Этот тип соответствует семантике определения пути из секции 2.1 согласно переходам модели Крипке.

Определим контекст

$$\Gamma_{path} = [path := path_{term} : path_{type}].$$

Формализация оператора AU в терминах *CIC*

Терм (обозначим его через AU_{term}), соответствующий оператору AU из CTL, который имеет тип (обозначим его тип через AU_{type})

$$(State \to Prop) \to (State \to Prop) \to State \to Prop,$$

устроен так:

$$\lambda(p : State \to Prop)(q : State \to Prop).$$

$$\lambda(w : State).\ \Pi(pi : nat \to State), (path\ pi) \to ((pi\ 0 : State) = (w : State)) \to$$

$$exists2(n : nat), (\Pi(m : nat), m < n \to p\ (pi\ m))\ \&\ q\ (pi\ n).$$

Если отображение $f : State \to Prop$ трактовать как множество состояний (всех термов st типа $State$, для которых верно $(f\ st) : Prop$), то, как можно видеть по структуре терма и устройству определения операции AU в разделе 2.1, возвращаемое отображение - это множество всех состояний модели, в которых выполняется формула $\phi AU \psi$, если p – множество всех состояний, в которых выполняется ϕ, а q – в которых выполняется ψ. Действительно, возвращаемое множество состояний удовлетворяет суждению о том, что для каждого состояния w этого множества и любого исходящего из w пути существует n-ое состояние пути на котором выполняется q, и на всех состояний с меньшим номером выполняется p.

Определим контекст

$$\Gamma_{AU} = [pAU := AU_{term} : AU_{type}].$$

Формализация оператора AR в терминах *CIC*

Терм (обозначим его через AR_{term}), соответствующий оператору AR из CTL, который имеет тип (обозначим его тип через AR_{type})

$$(State \to Prop) \to (State \to Prop) \to State \to Prop,$$

устроен так:

$$\lambda(p : State \to Prop)(q : State \to Prop).$$
$$\lambda(w : X).\ \Pi(pi : nat \to State), path\ pi \to ((pi\ 0 : State) = (w : State)) \to$$
$$\Pi(n : nat), (exists2\ m\ :\ nat, m < n\ \&\ p\ (pi\ m)) \vee q\ (pi\ n).$$

Если отображение $f : State \to Prop$ трактовать как множество состояний (всех термов st типа $State$, для которых верно $(f\ st)\ :\ Prop$), то, как можно видеть по структуре терма и устройству определения операции AR в разделе 2.1, возвращаемое отображение - это множество всех состояний модели, в которых выполняется формула $\phi AR\psi$, если p – множество всех состояний, в которых выполняется ϕ, а q – в которых выполняется ψ. Действительно, возвращаемое множество состояний удовлетворяет суждению о том, что для каждого состояния w этого множества и любого исходящего из w пути для любого n-ого состояния пути выполняется q, или существует состояние с меньшим номером, на котором выполняется p.

Определим контекст

$$\Gamma_{AR} = [pAR := AR_{term} : AR_{type}].$$

Формализация формулы *CTL* в терминах *CIC*

Для удобства описания формулы используется индуктивный тип $form$

$$
\begin{aligned}
&((form : Type), \{ \\
&\quad (fF \quad : form), \\
&\quad (fV \quad : nat \to form), \\
&\quad (fImp : form \to form \to form), \\
&\quad (fAnd : form \to form \to form), \\
&\quad (fOr \ : form \to form \to form), \\
&\quad (fAX \ : form \to form), \\
&\quad (fAU \ : form \to form \to form), \\
&\quad (fAR \ : form \to form \to form) \\
&\}).
\end{aligned}
$$

Конструкторы $fF, fV, fImp, fAnd, fOr, fAX, fAU, fAR$ индуктивного типа $form$ отвечают операциям $false, p, \to, \wedge, \vee, AX, AU, AR$. Термы типа $form$ естественным образом соответствуют формулам логики CTL.

4.1.3. Постановка задачи проверки модели

Преобразователь формулы в утверждение

Терм, отвечающий отношению выполнимости формулы на модели, устроен так:

$$
\begin{aligned}
&fixpoint\ satisfies(M : sts)(s : form)\{struct\ s\} : (state\ M) \to Prop :=\\
&(match\ s\ with\\
&\mid fF \qquad \Rightarrow \lambda(w : state\ M).\ False\\
&\mid fV\ v \qquad \Rightarrow \lambda(w : state\ M).\ label\ M\ v\ w\\
&\mid fImp\ p\ q \Rightarrow \lambda(w : state\ M).\ (satisfies\ M\ p\ w \to satisfies\ M\ q\ w)\\
&\mid fAnd\ p\ q \Rightarrow \lambda(w : state\ M).\ (satisfies\ M\ p\ w \wedge satisfies\ M\ q\ w)\\
&\mid fOr\ p\ q \quad \Rightarrow \lambda(w : state\ M).\ (satisfies\ M\ p\ w \vee satisfies\ M\ q\ w)\\
&\mid fAX\ p \quad \Rightarrow \lambda(w : state\ M).\ cAX(satisfies\ M\ p)\ w\\
&\mid fAR\ p\ q \ \Rightarrow \lambda(w : state\ M).\ pAR(satisfies\ M\ p)(satisfies\ M\ q)\ w\\
&\mid fAU\ p\ q \ \Rightarrow \lambda(w : state\ M).\ pAU(satisfies\ M\ p)(satisfies\ M\ q)\ w\\
&end : (state\ M) \to Prop).
\end{aligned}
$$

Обозначим этот терм через sat_{term}. Соответствующее отображение принимает на вход терм типа sts, соответствующий модели Крипке, и терм типа $form$, соответствующий формуле CTL, и возвращает предикат, который строится следующим образом: структура терма типа $form$ отвечает способу задания семантики формулы в разделе 2.1, структура подттерма $match$ отвечает пунктам задания семантики формул соответствующего оператора, и надтерм $fixpoint$ используется для индуктивного (рекурсивного) задания семантики в этих пунктах.

Определим контекст

$$\Gamma_{satisfies} = [satisfies := sat_{term} : \Pi(M : sts), form \to (state\ M) \to Prop].$$

Постановка задачи проверки модели

Пусть заданы контексты

$$\Gamma_{model} = [model := model_{term} : sts],$$

$$\Gamma_{formula} = [formula := formula_{term} : form],$$

где $model_{term}$,$formula_{term}$ – это термы, описывающие соответственно модель и формулу, для которых нужно проверить отношение выполнимости.

Задача проверки модели для логики *CTL* в терминах исчисления индуктивных конструкций – это проверка выводимости суждения о типе

$$\Gamma_{MC} \vdash t : Sat,$$

где $\Gamma_{MC} = \Gamma_{eq} \cup \Gamma_{or} \cup \Gamma_{and} \cup \Gamma_{ex} \cup \Gamma_{ex2} \cup \Gamma_{sts} \cup \Gamma_{form} \cup \Gamma_{model} \cup \Gamma_{formula} \cup \Gamma_{AR} \cup \Gamma_{AU} \cup \Gamma_{AX} \cup \Gamma_{satisfies}$, $Sat = \Pi(st : \ state\ model), ((init\ model)\ st) \to (satisfies\ model\ formula\ st)$.

4.2. Синтаксис и семантика языка тактик *Ltac*.

В *Coq* доступны средства для создания собственных тактик, которые можно применять для доказательства теорем. Одним из таких средств является язык тактик *Ltac*.

В работе [19], посвященной полному описанию синтаксиса и семантики, используемый математический аппарат является затруднительным для использования в данной работе. Документация же по языку *Ltac* [17] дает лишь не вполне строгое описание синтаксиса и семантики.

В данном разделе представлено описание части синтаксиса и семантики языка *Ltac*, потребовавшейся для решения задачи проверки модели в терминах исчисления индуктивных конструкций.

Часть синтаксиса языка тактик *Ltac* над множеством базовый тактик *Tactic*, множеством имен *Name*, множеством термов *term* можно задать следующей БНФ:

```
ltac_expr         ::=   ltac_expr ; ltac_expr
                      | [> ltac_expr | ... | ltac_expr ]
                      | (progress ltac_expr)
                      | (repeat ltac_expr)
                      | (try ltac_expr)
                      | let name in ltac_expr
                      | let name parameters := ltac_expr in ltac_expr
                      | (ltac_expr + ltac_expr)
                      | (tryif ltac_expr then ltac_expr else ltac_expr)
                      | (solve [ ltac_expr ])
                      | fail
                      | idtac
                      | tactic
                      | lazymatch type of name with
                        | term => ltac_expr
                        ...
                        | term => ltac_expr
```

```
  | name    => ltac_expr
  end
| lazymatch goal with
  | term => ltac_expr
  ...
  | term => ltac_expr
  | name    => ltac_expr
  end
| lazymatch term with
  | term => ltac_expr
  ...
  | term => ltac_expr
  | name    => ltac_expr
  end
```

где `tactic` $\in Tactic$, `name` $\in Name$, `term` $\in term$, `ltac_expr` - *Ltac*-выражение построенное на языке *Ltac*, `parameters`- набор параметров, где каждый параметр из множества $Tactic \bigcup Name \bigcup term \bigcup L$, где L – множество всех *Ltac*-выражений.

Связкой назовём запись a/l, где $a \in Name, l \in Tactic \bigcup Name \bigcup term \bigcup L$, и для такой связки будем называть l значением параметра a. Оценкой параметров назовём конечное множество связок $[a_1/l_1, ..., a_n/l_n]$, в котором каждый параметр $a_1, ...a_n \in Name$ в оценке параметров отлично от других имен в этой оценке, и имен базовых тактик. Запись $\Gamma \vdash A$, где Γ – это контекст, A – тип, будем называть целью. Цель $\Gamma \vdash A$ означает, что есть терм t типа A, для которого верно $\Gamma \vdash t : A$. Состоянием доказательства будем называть пару $((\Gamma \vdash A), \theta)$, где первый элемент – это цель, а второй – оценка параметров $\theta = [a_1/l_1, ..., a_n/l_n]$. Будем обозначать тот факт, что существует связка с параметром a_i значением l_i в оценке θ, записью вида $a_i/l_i \in \theta$.

Состояние управление — это *Ltac*-выражение. Состояние вычисления — это пара $(l|g)$, где l — состояние управления, а g — упорядоченный набор состояний доказательства.

Определим отношение переходов $\rightarrow$ на множестве состояний вычисления для *Ltac*-выражения l. Будем писать $S_1 \Rightarrow S_2$ для состояний вычисления S_1, S_2 в том случае, если существует последовательность $S_1 \rightarrow ... \rightarrow S_2$. Пусть l_1,l_2 – *Ltac*-выражения, g – набор состояний доказательства. Состояние ошибки – $\perp$.

1. Если $(l_i|(g_i)) \Rightarrow (idtac|(h_i)), \forall i = 1, ..., n$, то $([> l_1|..|l_n]|(g_1, ..., g_n)) \rightarrow (idtac|(h_1, .., h_n))$

   Если $\exists i = 1, ..., n$, $(l_i, (g_i)) \Rightarrow \perp$, $([> l_1|..|l_n]|(g_1, ..., g_n)) \rightarrow \perp$

2. Если $(l_1|g_1) \Rightarrow (idtac|g_2)$, то $(l_1; l_2|g_1) \rightarrow ([> l_2|...|l_2]|g_2)$.

   Если $(l_1|g_1) \Rightarrow \bot$, то $(l_1; l_2|g_1) \rightarrow \bot$.

3. Если $(l_1|g_1) \Rightarrow (idtac|g_1)$ или $(l_1|g_1) \Rightarrow \bot$, то $(repeatl_1|g_1) \rightarrow (idtac|g_1)$.

   Если $(l_1|g_1) \Rightarrow (idtac|g_2)$, где $g_1 \neq g_2$, то $(repeat\ l_1|g_1) \rightarrow (repeatl_1|g_2)$.

4. Если $(l_1|g_1) \Rightarrow (l_2|g_2)$, то $((try\ l_1)|g_1) \rightarrow ((try\ l_2)|g_2)$

   Если $(l_1|g_1) \Rightarrow \bot$, то $((try\ l_1)|g_1) \Rightarrow (idtac|g_1)$

5. Пусть $g_1 = ((h_1, \theta_1), ..., (h_m, \theta_m))$, тогда $(let\ name\ in\ l|g_1) \rightarrow (l|(h_1, k_1), ..., (h_m, k_m))$, где k_i получается из $\theta_i = [a_0/f_0, ..., a_s/f_s]$ следующим образом:

   если $\exists j, name = a_j$, то $k_i = [a_0/f_0, ..., a_j/name_{s+1}, ..., a_s/f_s]$,

   если $\forall j, name \neq a_j$, то $k_i = [a_0/f_0, a_s/f_s, name/name_{s+1}]$,

   где $name_{s+1} \in Name$ новое уникальное имя, которое не было использовано

6. Пусть $g_1 = ((h_1, \theta_1), ..., (h_m, \theta_m))$, тогда $(let\ rec\ name\ p_1, ..., p_n\ := l_1 in\ l_2|g_1) \rightarrow (l_2|(h_1, k_1), ..., (h_m, k_m))$, где k_i получается из $\theta_i = [a_0/f_0, ..., a_s/f_s]$ следующим образом:

   если $\exists j, name = a_j$, то $k_i = [a_0/f_0, ..., a_j/l_1(p_1, ..., p_n), ..., a_s/f_s]$,

   если $\forall j, name \neq a_j$, то $k_i = [a_0/f_0, a_s/f_s, name/(l_1\ p_1, ..., p_n)]$,

   где $p_1, ..., p_n, p_i \in term \bigcup Name \bigcup Tactic \bigcup L$ – это параметры l_1.

7. Если $(l_1|g_1) \Rightarrow (l_2|g_2)$, то $(tryif\ l_1\ then\ a\ else\ b|g_1) \rightarrow (tryif\ l_2\ then\ a\ else\ b|g_2)$

   Если $(l_1|g_1) \Rightarrow (idtac|g_2)$, то $(tryif\ l_1\ then\ a\ else\ b|g_1) \rightarrow (a|g_2)$

   Если $(l_1|g_1) \Rightarrow \bot$, то $(tryif\ l_1\ then\ a\ else\ b|g_1) \rightarrow (b|g_2)$

8. $(a + b, g) \rightarrow (tryif\ a\ then\ idtac\ else\ b|g)$

9. Если $(l_1|g_1) \Rightarrow (l_2|g_2)$, то $((solve\ l_1)|g_1) \rightarrow ((solve\ l_2)|g_2)$

   Если $(l_1|g_1) \Rightarrow (idtac|g_2)$ и $g_2 = ((\Gamma_1 \vdash, \theta_1), ..., (\Gamma_s \vdash, \theta_s))$ то $((solve\ l_1)|g_1) \rightarrow (idtac|g_2)$

   Если $(l_1|g_1) \Rightarrow (idtac|g_2)$ и $g_2 \neq ((\Gamma_1 \vdash, \theta_1), ..., (\Gamma_s \vdash, \theta_s))$ то $((solve\ l_1)|g_1) \rightarrow \bot$

10. $(fail|g) \to \bot$

11. Рассмотрим *Ltac*-выражение

$$
\begin{aligned}
&lazymatch\ type\ of\ name\ with\\
&|\ term_1(x_{1_1}, ..., x_{1_m}) \Rightarrow l_1\\
&\ ...\\
&|\ term_n(x_{n_1}, ..., x_{n_m}) \Rightarrow l_n\\
&|\ x \Rightarrow l_{n+1}\\
&end
\end{aligned}
$$

Пусть $g = (g_1, ..., g_n)$, где

$$g_i = ((\Gamma_i, name_i : \hat{term}_i(y_{i_1}, ..., y_{i_m}) \vdash A_i), [a_{i1}/f_{i1}, ..., name/name_i, ..., a_{im}/f_{im}])$$

Тогда

$$
\begin{aligned}
&(lazymatch\ type\ of\ name\ with\\
&|\ term_1(x_{1_1}, ..., x_{1_m}) \Rightarrow l_1\\
&\ ...\\
&|\ term_n(x_{n_1}, ..., x_{n_m}) \Rightarrow l_s\\
&|\ x \Rightarrow l_{s+1}\\
&end|g)
\end{aligned}
$$

$$\to ([o_1|...|o_n]|h),$$

где $o_i, i = 1, ..., n$ и $h = (h_1, ..., h_n)$ строятся следующим образом:

Для каждого терма $term_j$, если $term_j$ является уточнением терма $\hat{term}_i$, то $o_i = l_j$ и

$$h_i = ((\Gamma_i, name_i : \hat{term}_i(y_{i_1}, ..., y_{i_m}) \vdash A_i),$$

$$[a_{i1}/f_{i1}, ..., name/name_i, ..., a_{im}/f_{im}, x_{i_1}/y_{i_1}, ..., x_{i_m}/y_{i_m}])$$

Если $\exists i, j, i \neq j$: $term_i$ и $term_j$ одновременно являются уточнением для какого-то терма, то выбирается индекс с меньшим номером

Если $\forall j, term_j$ не является уточнением терма $\hat{term}_i$, то $o_i = l_{s+1}$ и

$$h_i = ((\Gamma_i, name_i : \hat{term}_i(y_{i_1}, ..., y_{i_m}) \vdash A_i),$$

$$[a_{i1}/f_{i1}, ..., name/name_i, ..., a_{im}/f_{im}, x/\hat{term}_i(y_{i_1}, ..., y_{i_m}))$$

Таким же образом работает тактика *lazymath goal with...* и *lazymath term with...*, только в этом случае, термы сопоставляются с целью текущего состояния доказательства и термами, соответственно.

Вычислением тактики l на наборе состояний доказательства $(g_1, ..., g_n)$ будем называть последовательность состояний вычислений, в которой соседние состояния соединены отношением $\rightarrow$ и последнее состояние, если оно есть, - это $(idtac|h)$, где h – некоторый набор состояний доказательства. Будем говорить, что тактика l строит успешный вывод для набора состояний доказательства, если вычисление этой тактики не содержит состояния ошибки и последнее состояние имеет следующий вид $(idtac|h)$, где $h = (h_1, ..., h_m)$, $h_i = ((\Gamma_i \vdash), \theta_i), i = 1, ..., m$.

4.3. Синтаксис и семантика базовых тактик

split. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash A_i \wedge B_i), \theta_i)$. Тактика *split* действует следующим образом:

$$(split|g) \rightarrow (idtac|h),$$

где $h = (h_1, \hat{h_1}, h_2, \hat{h_2}, ..., h_n, \hat{h_n})$, где $h_i = ((\Gamma \vdash A_i), \theta_i)$ и $\hat{h_i} = ((\Gamma \vdash B_i), \theta_i)$. Если $\exists i$ такой, что цель в состоянии доказательства g_i не является типом *and*, то

$$(split|g) \rightarrow \perp.$$

intro. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash \Pi(x : A_i)), B_i), \theta_i), H/\alpha \in \theta_i$. Тактика *intro* действует следующим образом:

$$(intro\ H|g) \rightarrow (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma, \alpha : A_i \vdash B_i\{x/\alpha\}), \theta_i)$. Если $\exists i$ такой, что цель в состоянии доказательства g_i не является типом вида $\Pi(x : A), B$, то

$$(intro\ H|g) \to \bot.$$

rewrite. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (\alpha : A_i = B_i) \vdash C_i), \theta_i)$, $H/\alpha \in \theta_i$. Если A_i свободно входит в тип C_i, то тактика *rewrite H* действует следующим образом:

$$(rewrite\ H|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = (\Gamma, (\alpha : A_i = B_i) \vdash C_i\{A_i/B_i\}), \theta_i), H/\alpha \in \theta_i$ иначе

$$(rewrite\ H|g) \to \bot.$$

Пусть $g_i = ((\Gamma, (\alpha : A_i = B_i), (\beta : D_i) \vdash C_i), \theta_i)$, $H_1/\alpha \in \theta_i, H_2/\beta \in \theta_i$.

1. Если A_i свободно входит в тип D_i, то тактика $rewrite\ H_1\ in\ H_2$ действует следующим образом:

   $$(rewrite\ H_1\ in\ H_2|g) \to (idtac|h),$$

   где $h = (h_1, ..., h_n)$, где $h_i = (\Gamma, (\alpha : A_i = B_i), (\beta : D_i\{A_i/B_i\}) \vdash C_i), \theta_i)$, иначе

   $$(rewrite\ H_1\ in\ H_2|g) \to \bot.$$

2. Если B_i свободно входит в тип D_i, то тактика $rewrite\ \leftarrow H_1\ in\ H_2$ действует следующим образом:

   $$(rewrite\ \leftarrow H_1\ in\ H_2|g) \to (idtac|h),$$

   где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma, (\alpha : A_i = B_i), (\beta : D_i\{B_i/A_i\}) \vdash C_i), \theta_i)$, иначе

   $$(rewrite\ \leftarrow H_1\ in\ H_2|g) \to \bot.$$

left и right. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma \vdash A_i \vee B_i), \theta_i)$. Тактика *left* действует следующим образом:

$$(left|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma_i \vdash A_i), \theta_i)$.

Тактика *right* действует следующим образом:

$$(right|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma_i \vdash B_i), \theta_i)$. Если $\exists i$ такой, что цель в состоянии доказательства g_i не является типом or, то

$$(left|g) \to \bot,$$

$$(right|g) \to \bot.$$

lia. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma \vdash B_i), \theta_i)$, где тип B_i является арфиметическим выражением, то есть тип построенный с помощью типов Nat, eq, le и других типов, отвечающих арифметическим и булевым операциям и отношениям. Тактика *lia* действует следующим образом:

$$(lia|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma_i \vdash), \theta_i)$. Если цель в состояниях g_i невозможно доказать, то

$$(lia|g) \to \bot.$$

discriminate. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (H : a = b) \vdash B_i), \theta_i)$, где тип B_i произвольная цель, a и b – это два неодинаковых конструктора одно и того же какого-то индуктивного типа. Тактика *discriminate* действует следующим образом:

$$(discriminate|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = (\Gamma_i \vdash, \theta_i)$. Если в текущем контексте не представлена гипотеза вида $(H : a = b)$, то

$$(discriminate|g) \to \bot.$$

compute. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma \vdash B_i), \theta_i)$, Тактика *compute* действует следующим образом:

$$(compute|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = (\Gamma_i \vdash_i, \theta_i),$, где C_i получается из B_i применением β и ι редукций.

Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (\alpha : A_i) \vdash B_i), \theta_i)$, $H/\alpha \in \theta_i$. Тактика *compute* действует следующим образом:

$$(compute\ in\ H|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma, (\alpha : C_i) \vdash B_i), \theta_i)$, где C_i получается из A_i применением β и ι редукций.

destruct.

1. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (\alpha : A_i \vee B_i) \vdash C_i), \theta_i)$, $H/\alpha \in \theta_i$. Тактика *destruct H as [H | H]* действует следующим образом:

   $$(destruct\ H\ as\ [H \mid H]|g) \to (idtac|h),$$

   где $h = (h_1, h_2, ..., h_n, \hat{h_1}, \hat{h_2}, ..., \hat{h_n})$, где $h_i = ((\Gamma, (\alpha : A_i) \vdash C_i), \theta_i)$ и $\hat{h_i} = ((\Gamma, (\alpha : B_i) \vdash C_i), \theta_i)$. Если $\exists i$ такой, что гипотеза α в состоянии доказательства не имеет тип *or*, то

   $$(destruct\ H\ as\ [H \mid H]|g) \to \bot.$$

2. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (\alpha : A_i \wedge B_i) \vdash C_i), \theta_i)$, $H/\alpha, a/\beta, b/\gamma \in \theta_i$. Тактика *destruct H as (a & b)* действует следующим образом:

   $$(destruct\ H\ as\ (a\&b)|g) \to (idtac|h),$$

   где $h = (h_1, h_2, ..., h_n)$, где $h_i = ((\Gamma, (\beta : A_i), (\gamma : B_i) \vdash C_i), \theta_i)$ Если $\exists i$ такой, что гипотеза α в состоянии доказательства не имеет тип *and*, то

   $$(destruct\ H\ as\ (a\&b)|g) \to \bot.$$

3. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma, (\alpha : A) \vdash C_i), \theta_i)$, $H/\alpha \in \theta_i$, где A–индуктивный тип. Тактика *destruct H* действует следующим образом:

   $$(destruct\ H|g) \to (idtac|h),$$

   где h равен конкатенации наборов G_i, $G_i = (g_{i1}, ..., g_{in})$, где g_{is} получается из g_i следующим образом: каждое вхождение гипотезы α в типы остальных гипотез заменяется на s-ый конструктор индуктивного типа A, иначе если A не является индуктивным типом, то

   $$(destruct\ H|g) \to \bot.$$

apply. Пусть $g_i = ((\Gamma, (\alpha : D_i \to B_i)(\beta : C_i) \vdash D_i), \theta_i)$, $H_1/\alpha H_2/\beta \in \theta_i$, $D_i = C_i$. Тактика *apply* H_1 *in* H_2 действует следующим образом:

$$(apply\ H_1\ in\ H_2|g) \to (idtac|h),$$

где $h = (h_1, h_2, ..., h_n)$, где $h_i = ((\Gamma, (\alpha : D_i \to B_i)(\beta : B_i) \vdash D_i), \theta_i)$, если $D_i \neq C_i$, то

$$(apply\ H_1\ in\ H_2|g) \to \bot.$$

Пусть $g_i = ((\Gamma, (\alpha : D_i) \vdash C_i), \theta_i)$, $H/\alpha \in \theta_i$, $D_i = C_i$. Тактика *apply H* действует следующим образом:

$$(apply\ H_1\ in\ H_2|g) \to (idtac|h),$$

где $h = (h_1, h_2, ..., h_n)$, где $h_i = ((\Gamma, (\alpha : D_i) \vdash), \theta_i)$, если $D_i \neq C_i$, то

$$(apply\ H_1\ in\ H_2|g) \to \bot.$$

pose proof. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash C_i), \theta_i)$, $H/\alpha \in \theta_i$. Тактика *pose proof (t : A) as H* действует следующим образом:

$$(pose\ proof\ (t : A)\ as\ H|g) \to (idtac|h),$$

где $h = (h_1, h_2, ..., h_n)$, $h_i = ((\Gamma_i, (\alpha : A) \vdash C_i), \theta_i)$,где t– корректно построенный в контексте Γ_i терм типа A

eexists. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash exists2(k : A), B_i \& C_i), \theta_i)$. Тактика *eexists t* действует следующим образом:

$$(eexists\ t|g) \to (idtac|h),$$

где $h = (h_1, \hat{h_1}, h_2, \hat{h_2}, ..., h_n, \hat{h_n})$, где $h_i = ((\Gamma_i \vdash B_i\{k/t\}), \theta_i)$ и $\hat{h_i} = ((\Gamma_i \vdash C_i\{k/t\}), \theta_i)$, где t– корректно построенный в контексте Γ_i терм типа A.

assert. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash B_i), \theta_i)$. Тактика *assert (A)* действует следующим образом:

$$(assert(A)|g) \to (idtac|h),$$

где $h = (\hat{g_1}, g_1, \hat{g_2}, g_2, ..., \hat{g_2}, g_n), \hat{g_i} = ((\Gamma_i \vdash A), \theta_i)$, где A– корректно построенный тип в контексте Γ_i .

Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i, (\beta : A_i) \vdash B_i), \theta_i), H_1/\alpha, H_2/\beta \in \theta_i$. Тактика *assert* $(H_1 := H_2)$ действует следующим образом:

$$(assert(H_1 := H_2)|g) \to (idtac|h),$$

где $h = (h_1, h_2, ..., h_n)$, где $h_i = ((\Gamma_i, (\beta : A_i), (\alpha : A_i) \vdash C_i), \theta_i)$.

Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash B_i), \theta_i), H/\alpha$. Тактика *assert (H: A)* действует следующим образом:

$$(assert(H : A)|g) \to (idtac|h),$$

где $h = (h_1, \hat{g_1}, g_2, \hat{g_2}, ..., h_n, \hat{g_n}), \hat{g_i} = ((\Gamma_i, (\alpha : A) \vdash B_i), \theta_i), h_i = ((\Gamma_i \vdash A), \theta_i)$, где A– корректно построенный тип в контексте Γ_i .

reflexivity. Пусть $g = (g_1, ..., g_n)$, где $g_i = ((\Gamma_i \vdash B_i = B_i), \theta_i)$. Тактика *reflexivity* действует следующим образом:

$$(reflexivity|g) \to (idtac|h),$$

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma_i \vdash), \theta_i)$. Если цель в состояниях доказательства не является индуктивным типом eq, то

$$(reflexivity|g) \to \bot.$$

4.4. Ограничения на представление модели Крипке

В типе *sts* моделей Крипке не накладывается требование конечности множества состояний модели, обычное для моделей Крипке и содержащееся в определении в разделе 2.1. Чтобы соблюсти это требование и конкретизировать тип моделей Крипке, будем считать, что состояния модели задаются индуктивным типом $((A : Type), \{(constr_1 : A), ..., (constr_n : A)\})$, то есть состояниями являются термы $constr_1$, ..., $constr_n$ и только они.

Для представления отношения переходов введём вспомогательные индуктивные типы $prod$ и $list$, которые соответственно описывают пары и списки термов:

$$(prod(A : Type)(B : Type) : Type, \{pair : A \to B \to (prod\ A\ B)\}),$$

$$(list(A : Type) : Type, \{nil : list\ A, cons : A \to list\ A \to list\ A\}).$$

Множество переходов модели Крипке представим в виде списка смежности - списка пар вида $(r_0, (r_1, ..., r_k))$ с попарно различными элементами r_0, где r_0 - состояние, из которого исходит переход, а $r_1, ..., r_k$ - все состояния, в которые ведут переходы изr_0. Таким образом, список смежности для переходов модели Крипке задаётся термом типа $list\ (prod\ A\ (list\ A))$.

Отношение переходов модели Крипке, основанное на списке смежности l, задаётся термом $(\lambda(s_1 : A), \lambda(s_2 : A), make\_prop\ s_1\ s_2\ t)$ типа $A \to A \to Prop$, где

$fixpoint\ make\_dis(A : Type)(s_2 : A)(states\_in : listA)\{structstates\_in\} : Prop :=$

$(match\ states\_in\ with$

$|\ cons\ head\ nil \Rightarrow eq\ s_2\ head$

$|\ cons\ head\ tail \Rightarrow or(eq\ s_2\ head)(make\_dis\ A\ s_2\ tail)$

$|\ nil \Rightarrow False$

$end : \Pi(A : Type), A \to listA \to Prop),$

$fixpoint\ make\_prop(A : Type)(B : Type)(s_1 : A)(s_2 : B)$

$(list\_conn : list(prod\ A\ (list\ B)))\{struct\ list\_conn\} : Prop :=$

$(match\ list\_conn\ with$

$|\ cons\ (pair\ b_1\ b_2)\ nil => ((eq\ s_1\ b_1) \to (make\_dis\ s_2\ b_2))$

$|\ cons\ (pair\ b_1\ b_2)\ tail \Rightarrow and((eq\ s_1\ b_1) \to (make\_dis\ s_2\ b_2))(make\_prop\ s_1\ s_2\ tail)$

$|\ nil \Rightarrow False$

$end : \Pi(A : Type), A \to listA \to Prop).$

С поправкой на бета и йота редукцию, этот терм можно записать в виде

$$(\lambda(s_1 : A), \lambda(s_2 : A), (s_1 = constr_1 \to (s_2 = constr_{11} \vee ... \vee s_2 = constr_{1m})) \wedge ...$$

$$... \wedge (s_1 = constr_n \to (s_2 = constr_{n1} \vee ... \vee s_2 = constr_{nm}))$$

явного перечисления всех дуг $(constr_i, constr_{ij})$.

Аналогично можно задать список l пар $(r_0, (i_1, ..., i_s))$ с попарно различными элементами r_0, где r_0 - состояние, а $i_1, ..., i_s$ термы типа Nat, соответствующие переменным

$x_{i_1}, ..., x_{i_n}$, и терм $(\lambda(a : nat), \lambda(b : A), make\_prop\ b\ a\ l)$ типа $Nat \to A \to Prop$, с поправкой на β и ι редукцию записывающийся в виде

$$(b = constr_1 \to (a = k_{11} \vee ... \vee a = k_{1m})) \wedge ...$$

$$... \wedge (b = constr_n \to (a = k_{n1} \vee ... \vee a = k_{nm}))$$

и естественно задающий функцию разметки модели Крипке, согласно которой состояние $constr_i$ помечено множеством атомарных высказываний $\{x_{k_{i1}}, ..., x_{k_{im}}\}$.

Для технической простоты будем рассматривать только модели с одним начальным состоянием, и тот факт, что состояние v является начальным, будем обозначать термом $(\lambda(s : A),\ eq\ s\ v)$ типа $A \to Prop$.

На способ записи доказательства тотальности отношения переходов в типе sts не будем накладывать никаких ограничений.

4.5. Алгоритмы (тактики)

Далее полагаем, что в контексте содержится запись $model := t : sts$, где t - терм типа sts, удовлетворяющий ограничениям раздела 4.4. В разделах 4.5.1-4.5.6 описываются тактики, доказывающие выполнимость формулы конкретного вида (с конкретной внешней операцией), и в них используется центральная вспомогательная тактика $applicator\ n\ f$, доказывающая выполнимость формулы произвольного вида, отвечающей терму f типа $form$, для модели Крипке с n состояниями. В разделе 4.5.7 описывается главная тактика $solver$, предназначенная для автоматического доказательства теоремы такого вида, как требуется в формальной постановке задачи в разделе 4.1, при помощи выполнения тактики $applicator$.

4.5.1. solve_fV

Тактика для доказательства выполнимости формулы вида $fV\ n$, где n - терм типа nat (то есть для переменной x_n), имеет следующий вид:

```
compute;
repeat split;
let pre in
intro pre;
( rewrite init_ in pre; discriminate )
```

```
+
( repeat ( (left; reflexivity) + (right; reflexivity) + right) )
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где g_1 имеет следующий вид

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ (fV\ n)\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметром алгоритма является имя гипотезы α типа $\beta = v$, где $\alpha \in Name$, v – терм типа *state model*, β - терм типа *state model*, построенный в контексте Γ. Тогда выполняются следующие действия:

1. При выполнении *compute* цель в состоянии доказательства принимает вид

$$(\beta = constr_1 \to (n = n_{11} \vee ... \vee n = n_{1m})) \wedge ... \wedge (\beta = constr_k \to (n = n_{s1} \vee ... \vee n = n_{sm}))$$

2. Выполнение *repeat split* разбивает состояние доказательства на k состояния вида

$$((\Gamma, (\alpha : \beta = v) \vdash (\beta = constr_j) \to (n = n_{j1} \vee ... \vee n = n_{jm})), \theta).$$

3. Выполнение *intro pre* преобразует все состояния к виду

$$((\Gamma, (\alpha : \beta = v)(\gamma : \beta = constr_j) \vdash (n = n_{j1} \vee ... \vee n = n_{jm})), \theta).$$

4.1. Если термы v и $constr_j$ не совпадают, то после выполнения *rewrite init_ in pre*; *discriminate*, строится гипотеза о равенстве двух неравных термах, и выполнение *discriminate* завершает доказательство.

4.2. Иначе при помощи $repeat((left; reflexivity) + (right; reflexivity) + right)$ перебираются равенства под дизъюнкцией в правой части цели, пока не будет найдено такое, в котором знаком равенства соединены одинаковые числа (термы типа *nat*), и выполнение $reflexivity$ завершает доказательство.

4.5.2. solve_fOr

Тактика для доказательства выполнимости формулы вида $(fOr\ f_1\ f_2)$ (то есть для формулы вида $f_1 \vee f_2$), где f_1, f_2 - подтермы типа $form$, имеет следующий вид:

```
compute;
(left; solve [ tac1 init_ ]) + (right; solve [ tac2 init_ ])
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где g_1 имеет следующий вид

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ (fOr\ f_1\ f_2)\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметрами алгоритма являются имя гипотезы α типа $\beta = v$, где $\alpha \in Name$, v – терм типа *state model*, β - терм типа *state model*, построенный в контексте Γ. Положим $tac\_1 = applicator\ n\ f_1, tac\_2 = applicator\ n\ f_2$, которые соответственно доказывают выполнимость формул, отвечающие термам f_1 и f_2. Параметром этих тактик является имя гипотезы, тип которой имеет вид $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа *state model*, $\hat{\beta}$ - терм типа *state model*, построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. При выполнении *compute* цель в состояниях доказательства принимает вид

$$(satisfies\ model\ (f_1)\ \beta) \vee (satisfies\ model\ (f_2)\ \beta).$$

2. Выполнение ($left$; $solve$ [$tac1\ init\_$]) преобразует цель в состоянии доказательства к виду ($satisfies\ model\ (f_1)\ \beta$) и далее применяется тактика $tac1$ с параметром α.

3.1 Если тактика $tac1$ строит успешный вывод, то доказательство завершено.

3.2 Если тактика $tac1$ не строит успешный вывод, то цель вновь имеет вид ($satisfies\ model\ (f_1)\ \beta$) $\vee$ ($satisfies\ model\ (f_2)\ \beta$), и применяется тактика ($right$; $solve$ [$tac2\ init\_$]), которая преобразует цель в состоянии доказательства к виду ($satisfies\ model\ (f_2)\ \beta$) и выполняется тактика $tac2$ с параметром α. Если тактика $tac2$ строит успешный вывод для правой части дизъюнкции, то доказательство завершено, иначе $\perp$.

4.5.3. solve_fAnd

Тактика для доказательства выполнимости формулы вида ($fAnd\ f_1\ f_2$) (то есть для формулы вида $f_1 \wedge f_2$), где f_1, f_2 - подтермы типа $form$, имеет следующий вид:

```
compute;
split; [> solve [ tac1 init_ ] | solve [ tac2 init_] ].
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где g_1 имеет следующий вид

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ (fAnd\ f_1\ f_2)\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметрами алгоритма являются имя гипотезы α типа $\beta = v$, где $\alpha \in Name$, v – терм типа *state model*, β - терм типа *state model*, построенный в контексте Γ. Положим $tac\_1 = applicator\ n\ f_1, tac\_2 = applicator\ n\ f_2$, которые соответственно доказывают выполнимость формул, отвечающие термам f_1 и f_2. Параметром этих тактик является имя гипотезы, тип которой имеет вид $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа *state model*, $\hat{\beta}$ - терм типа *state model*, построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. При выполнении *compute* цель в состояниях доказательства принимает вид

$$(satisfies\ model\ (f_1)\ \beta) \wedge (satisfies\ model\ (f_2)\ \beta).$$

2. Применяется тактика *split*, которая «разбивает» состояние доказательства на два состояния вида

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ f_1\ \beta)), \theta)$$

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ f_2\ \beta)), \theta)$$

3. Для первого состояния с параметром α применяется тактика $tac1$, а для второго – тактика $tac2$. Если обе тактики строят успешный вывод, то доказательство завершено, иначе $\perp$.

4.5.4. solve_fAX

Тактика для доказательства выполнимости формулы вида $(fAX\ f_1)$ (то есть для формулы вида $AX\ f_1$), где f_1 - подтерм типа $form$, имеет следующий вид:

```
compute;
let next_state in intro next_state;
let trans_to_next_state in intro trans_to_next_state;
compute in trans_to_next_state;
let rec loop conj_hyp init_ :=
  lazymatch type of conj_hyp with
  | and l r =>
    let a in let b in
    destruct conj_hyp as (a & b);
    (apply a in init_) + (apply b in init_) + (loop b)
  | r =>
    idtac
end in
loop trans_to_next_state init_;
  repeat (
    lazymatch type of init_ with
    | or l r =>
      destruct init_ as [init_ | init_]
    | r =>
      idtac
    end
);
solve [ tac1 init_ ]
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где g_1 имеет следующий вид

$$((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ (fAX\ f_1)\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметрами алгоритма являются имя гипотезы α типа $\beta = v$, где $\alpha \in Name$, v – терм типа *state model*, β - терм типа *state model*, построенный в контексте Γ. Положим $tac\_1 = applicator\ n\ f_1$, который доказывает выполнимость формулы, отвечающей терму f_1. Параметром этой тактики является имя гипотезы, тип которой имеет вид $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа *state model*, $\hat{\beta}$ - терм типа *state model*, построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. При выполнении тактик *compute* и *intro* в состоянии доказательства применяются правила редукции к цели и предположения «переносятся в гипотезы». Состояние доказательства на данном этапе имеет следующий вид

$$((\Gamma, (\alpha : \beta = v), (\gamma : state\ model), (\delta : (trans\ model)\ \beta\ \gamma)$$

$$\vdash (satisfies\ model\ f\ \gamma)), \theta),$$

$$next\_state/\gamma, trans\_to\_next\_state/\delta \in \theta.$$

2. При выполнении тактики *compute in trans_to_next_state* тип гипотезы δ будет иметь вид

$$(\beta = constr_1 \to (\gamma = u_{1_1} \vee ... \vee \gamma = u_{1_m})) \wedge ... \wedge (\beta = constr_n \to (\gamma = u_{n_1} \vee ... \vee \gamma = u_{n_m}))$$

3. Так как по условию в каждой скобке слева от имликации присутствует каждый конструктор, и если v является конструктором $constr_j$, то подвыражение *loop* будет «перебирать» каждую скобку до тех пор пока слева от импликации не появится тип $\beta = contr_j$, и далее выполняется тактика *apply*, которая преобразует тип гипотезы α в $(\gamma = u_{j_1} \vee ... \vee \gamma = u_{j_m})$.

5. Далее с помощью тактик *repeat* и *lazymatch* состояние доказательства «разбивается» на jm состояний вида

$$((\Gamma, (\alpha : (\gamma = u_{j_s})(\gamma : state\ model)(\beta : (trans\ model)\ \kappa\ \alpha)$$

$$\vdash (satisfies\ model\ f_1\ \gamma)), \theta), s = 1, ..., j_m.$$

6. Далее для каждого из j_m состояний выполняется тактика $tac1$ с параметром которой является α.

4.5.5. solve_fAU

Рассмотрим вспомогательные тактики.

unsplit. Это тактика, которая из двух гипотез создает третью гипотезу, которая является конъюнкцией типов первых двух гипотез. Параметрами тактики являются $H1,\ H2$– имена гипотез, типы которых будут использоваться для построения конъюнкции, $H12$ – имя гипотезы, тип которой будет конъюнкция типов гипотез $H1,\ H2$.

```
lazymatch type of H1 with
| t1 =>
  lazymatch type of H2 with
  | t2 =>
    assert (H12: t1 /\ t2);
    [
```

```
      split; [ apply H1 | apply H2 ]
      |
      idtac
    ]
  end
end
```

next_state_gen. Тактика, которая создает гипотезу о состоянии в пути в следующий момент времени при помощи гипотезы о состоянии в пути в настоящий момент времени, имеет следующий вид:

```
let is_path_pi_i in
pose proof (is_path_pi i) as is_path_pi_i;
compute in is_path_pi_i;
let rec loop conj_hyp init_ :=
lazymatch type of conj_hyp with
| and l r =>
    let a in let b in
    destruct conj_hyp as (a & b);
    (apply a in init_) + (apply b in init_) + (loop b)
| r =>
    idtac
end
in
loop is_path_pi_j first_state;
repeat (
    lazymatch type of first_state with
    | or l r  =>
        destruct first_state as [first_state | first_state]
    | r =>
        idtac
    end
)
```

Эта тактика отвечает следующему алгоритму. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (\beta : \Pi(n : nat), (trans\ model)(pi\ n)(pi(S\ n))),$$

$$(pi : nat \to (state\ model)), (\gamma : pi\ j = v_i) \vdash A), \theta_i),$$

$$is\_path\_pi/\beta, first\_state/\gamma \in \theta.$$

Параметрами тактики являются терм i типа nat, гипотеза is_path_pi типа $\Pi(n : nat), (trans\ model)(pi\ n)(pi(S\ n))$, гипотеза $first\_state$ типа $pi\ i = v_i$, где v_i – это один

из конструкторов типа $(state\ model)$, pi имеет тип $nat \to (state\ model)$. Данная тактика работает только с гипотезами.

Тогда выполняются следующие действия:

1. Выполняется тактика $pose\ proof$, которая вносит новую гипотезу $is\_path\_pi\_i$ в контекст, которая имеет тип $(trans\ model)(pi\ i)(pi(S\ i))$.

2. При выполнении тактики $compute\ in\ is\_path\_pi\_i$ тип гипотезы принимает вид

$$(pi\ i = constr_1 \to (pi(S\ i) = u_{11} \vee ... \vee pi(S\ i) = u_{1m})) \wedge ...$$

$$... \wedge (pi\ i = constr_n \to (pi(S\ i) = u_{n1} \vee ... \vee pi(S\ i) = u_{nm}))$$

3. Далее схожим образом, как в алгоритме $solve\_fAX$ строятся s_m состояний доказательств для каждого состояния g_s

$$((\Gamma_s, (\alpha : pi(S\ i) = u_{sk}) \vdash A, k = 1, ..., s_m.$$

Определим терм, который интепретируется как усеченная разность между термом m типа nat и термом $(S\ O)$ (1). И будем обозначать $m - 1$.

$$(\lambda(m : nat), match\ m\ with\ |S\ n \Rightarrow n|O \Rightarrow O\ end)$$

Далее будем считать построенными следующие термы t_1, t_2, t_3, t_4, и по умолчанию будем считать контекст $\hat{\Gamma}$ включенным в каждый.

$$\hat{\Gamma} =$$

$$(usefull := t_1 : \Pi(m : nat), \Pi(n : nat), S\ m \leq n \to S\ m = n \vee S\ m \leq n - 1,$$

$$usefull2 := t_2 : \Pi(m : nat), \Pi(n : nat), S\ m = S\ n \to m = n,$$

$$usefull3 := t_3 : \Pi(m : nat), \Pi(n : nat), m \leq n \to m - 1 \leq n - 1,$$

$$usefull4 := t_4 : \Pi(m : nat), \Pi(n : nat), S\ m = n \to m = n - 1),$$

где $\leq$ – это обозначение для индуктивного типа le, и $m-1$, $n-1$ – это усеченная разность с единицей.

proof_ex_sat. Тактика, которая для префикса конкретного пути строит вывод для цели вида $p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)\ pi$, где pi типа $nat \to (state\ model)$, имеет следующий вид:

```
compute;
eexists i;
[
    let m in intro m;
    let lt_m in intro lt_m;
    let a in let b in
    destruct seq as (a & b);
    let rec loop i sequence lt_m :=
        lazymatch type of sequence with
        | and l r =>
            let head in
            let tail in
            destruct sequence as (head & tail);
            apply usefull in lt_m;
            compute in lt_m;
            destruct lt_m as [lt_m| lt_m];
            [
              apply usefull4 in lt_m; compute in lt_m ;rewrite lt_m;
              tac1 head
              |
              (loop (i-1) tail lt_m)
            ]
        | r =>
            apply usefull in lt_m;
            destruct lt_m as [lt_m| lt_m];
            compute in lt_m;
            [
            apply usefull4 in lt_m;
            compute in lt_m ;
            rewrite lt_m;
            tac1 sequence
            |
            lia
            ]
        end in
    loop i b lt_m
|
    let a in
    let b in
    destruct seq as (a&b);
    tac2 a
].
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на

вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (pi : nat \to (state\ model)), (\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0)$$

$$\vdash p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)\ pi), \theta_i), seq/\delta \in \theta.$$

Параметром алгоритма являются терм i типа nat, гипотеза seq типа $pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0$. Положим $tac\_1 = applicator\ n\ f_1, tac\_2 = applicator\ n\ f_2$, которые соответственно доказывают выполнимость формул, отвечающие термам f_1 и f_2. Параметром этих тактик является имя гипотезы, тип которой имеет вид $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа $state\ model$, $\hat{\beta}$ - терм типа $state\ model$, построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. Выполняется тактика $compute$; $eexists\ i$, и состояние доказательства «разбивается» на два состояния, которые соответственно имеют цели $\Pi(m : nat), m < i \to satisfies\ model\ f_1\ (pi\ m)$ и $satisfies\ model\ f_2\ (pi\ i)$.

2.1 Рассмотрим первое состояние

$$((\Gamma, (\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0) \vdash \Pi(m : nat), m < i \to satisfies\ model\ f_1\ (pi\ m)), \theta),$$

$$seq/\delta \in \theta_i,$$

которое после выполнения тактик $intro$ и $destruct$ принимает вид

$$((\Gamma, (\phi : pi\ i = v_i), (\hat{\delta} : pi\ (i-1) = v_{i-1} \wedge ... \wedge pi\ 0 = v_0), (\kappa : nat)(\gamma : \kappa < i) \vdash$$

$$satisfies\ model\ f_1\ (pi\ \kappa)), \theta), a/\phi, b/\hat{\delta}, m/\kappa, lt\_m/\gamma \in \theta_i$$

2.2 Выполняется тактика $lazymatch$, которая в зависимости от типа $\hat{\delta}$ выполняет соответствующую тактику.

2.2.1 Если тип есть and, то гипотеза $(\hat{\delta} : pi\ (i-1) = v_{i-1} \wedge ... \wedge pi\ 0 = v_0)$ снова при помощи тактики $destruct$ «разбивается» на две гипотезы $(\hat{\delta_1} : pi\ (i-2) = v_{i-2} \wedge ... \wedge pi\ 0 = v_0)$ и $(\hat{\delta_2} : pi\ (i-1) = v_{i-1})$ в текущем состоянии доказательства.

2.2.2 Далее выполняется тактик $apply$, которая применяет гипотезу $usefull$ к гипотезе γ, и ее тип принимает вид $S\ \kappa = i\ \vee\ \kappa < (i-1)$

2.2.3 Затем тактика *destruct lt_m as* $[lt\_m|lt\_m]$ «разбивает» состояние доказательства на два состояния с гипотезами типа $S\ \kappa = i$ и $\kappa < (i-1)$. Для первого состояния гипотеза типа $S\ \kappa = i$ при помощи тактики *apply* и гипотезы $usefull4$ преобразуется в $\kappa = i - 1$, выполняется тактика $tac1$ с параметром $\hat{\delta_2}$ для цели этого состояния $satisfies\ model\ f_1\ (pi\ \kappa)$. Для второго состояния рекурсивно выполняется тактика *loop* с гипотезой $\hat{\delta_1}$ в качестве параметра.

2.3 Если тип гипотезы $\hat{\delta}$ не является типом *and*, то тип $\hat{\delta}$ – это $pi\ 0 = v_0$, а тип гипотезы γ – это $i < 1$

2.3.1 Далее выполняется тактика *apply*, которая применяет гипотезу $usefull$ к гипотезе γ, и ее тип принимает вид $S\ \kappa = 1\ \vee\ \kappa < 0$.

2.3.2 Затем тактика *destruct lt_m as* $[lt\_m|lt\_m]$ «разбивает» состояние доказательства на два состояния с гипотезами типа $S\ \kappa = 1$ и $\kappa < 0$. Для первого состояния гипотеза типа $S\ \kappa = 1$ при помощи тактики *apply* и гипотезы $usefull4$ преобразуется в $\kappa = 0$, выполняется тактика $tac1$ с параметром $\hat{\delta_2}$ для цели этого состояния $satisfies\ model\ f_1\ (pi\ \kappa)$. Для второго состояния выполняется тактика *lia*, так как можно построить невозможное неравенство $\kappa < 0$, что позволяет доказать любую цель, включая текущую.

3.1 Рассмотрим второе состояние из пункта 1: $((\Gamma, (\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0) \vdash satisfies\ model\ f_2\ (pi\ i)), \theta_i)$

3.2. Выполняется тактика *destruct*, которая «разбивает» гипотезу δ на гипотезы $(\phi : pi\ i = v_i), (\hat{\delta} : pi\ (i-1) = v_{i-1} \wedge ... \wedge pi\ 0 = v_0)$.

3.3. Выполняется тактика $tac2$ с параметром ϕ.

loop1. Тактика, которая для каждого возможного пути строит конечный префикс и доказывает для него цель вида $p\_until(state\ \ model)(satisfies\ \ model\ \ f_1)(satisfies\ \ model\ \ f_2)\ \ pi$, где pi типа $nat \to (state\ model)$, имеет следующий вид:

```
tryif
    assert (i=n);[solve [ lia ] | idtac])
then
```

```
    fail
else
    let new in
    assert(new:=last_stop);
    next_state_gen i is_path_pi new;
    let H in
    unsplit new prev_acc H;
    (solve [(proof_ex_sat i prev_acc tac1 tac2)])
    +
    (loop1 (S i) n is_path_pi new H tac1 tac2)
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (pi : nat \to (state\ model)), (\omega : path(state\ model)(trans\ model)\pi), (\alpha : pi\ i = v_i),$$

$$(\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0) \vdash p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)\ pi), \theta),$$

$$prev\_acc/\delta, last\_stop/\alpha, is\_path\_pi/\omega \in \theta.$$

Параметром алгоритма являются терм i типа nat, терм n типа nat (количество конструкторов индуктивного типа $(state\ model)$), гипотеза $last\_stop$ типа $pi\ i = v_i$, гипотеза $prev\_acc$ типа $pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0$ и тактики $tac1, tac2$, которые соответственно доказывают выполнимость формул вида f_1 и f_2. Параметром тактик $tac1, tac2$ является имя гипотезы типа $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа $state\ model$, $\hat{\beta}$ - терм типа $state\ model$, построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. Выполняется тактика $tryif$, которая «проверяет» равенство $i = n$ так: выполняется тактика $assert$, которая создает новое состояние доказательства с целью $i = n$, а затем выполняется тактика lia, которая строит доказательство для этой цели. Если lia строит успешный вывод, то $\perp$.

2. Если lia не строит успешный вывод, то выполняется тактика $assert$, которая создает новую гипотезу α и состояние доказательства принимает вид

$$((\Gamma, (\beta : pi\ i = v_i), (\alpha : pi\ i = v_i), (\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0) \vdash$$

$$p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)pi), \theta),$$

$$new/\beta \in \theta$$

3. Выполняется вспомогательная тактика $next\_state\_gen$ с параметрами i, ω, α, и состояние доказательства «разбивается» на m состояний, где тип гипотезы α имеет вид $\alpha : pi\ (S\ i) = \hat{v_j}, j = 1, ..., m$.

4. Выполняется тактика $unsplit$ с параметрами α, δ, которая вносит новую гипотезу $\hat{\delta}$ типа $pi\ (S\ i) = \hat{v_i} \wedge pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0$ в контекст.

5. Далее выполняется вспомогательная тактика $proof\_ex\_sat$ с параметрами $i, \hat{\delta}, tac1, tac2$. Если тактика $proof\_ex\_sat$ строит успешный вывод для данного «префикса» $\hat{\delta}$, то доказательство завершается.

6. Если тактика $proof\_ex\_sat$ не строит успешный вывод для данного «префикса», то выполняется тактика $loop1$ с параметрами $(S\ i), n, \omega, \alpha, \hat{\delta}, tac1, tac2$.

solve_fAU. Тактика для доказательства выполнимости формулы вида $(fAU\ f_1\ f_2)$ (то есть для формулы вида $AU\ f_1\ f_2$), где f_1, f_2 - подтермы типа $form$, имеет следующий вид:

```
let pi in
intro pi;
let is_path_pi in
intro is_path_pi;
let first_state in
intro first_state;
rewrite init_l in first_state;
(eexists 0; [> lia | solve [tac2 first_state] ])
+
(loop1 0 n is_path_pi first_state first_state tac1 tac2)
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ (fAU\ f_1\ f_2)\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметрами алгоритма являются терм n типа nat (количество состояний в модели), имя гипотезы α типа $\beta = v$, где $\alpha \in Name$, v – терм типа $state\ model$, β - терм типа $state\ model$, построенный в контексте Γ, тактики $tac1, tac2$, которые соответственно доказывают выполнимость формул вида f_1 и f_2. Параметром тактик $tac1, tac2$ является имя гипотезы типа $\hat{\beta} = \hat{v}$, где $\hat{v}$ – терм типа $state\ model$, $\hat{\beta}$ - терм типа $state\ model$,

построенный в контексте того, состояния доказательства для которого применяется тактика.

Тогда выполняются следующие действия:

1. Выполняются тактики *intro* и *rewrite*, и состояние доказательства принимает вид

$$((\Gamma, (\beta : \alpha = v), (\delta : pi\ 0 = v),$$

$$(pi : nat \to (state\ model)), (\gamma : path(state\ model)(trans\ model)\ pi)$$

$$\vdash p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)\pi), \theta_i),$$

$$is\_path\_pi/\gamma, first\_state/\delta \in \theta.$$

2. Выполняется тактика ($eexists$ 0; [> lia | $solve$ [$tac2$ $first\_state$]]) для доказательства выполнимости формулы для пути состоящего из одного состояния $pi\ 0 = v$. Согласно семантике тактики *eexists* состояние «разбивается» на два состояния. Для первого состояния цель доказывается с помощью противоречия $0 < 0$ и тактики *lia*. Для второго состояния применяется тактика $tac2$ с параметром δ, если тактика $tac2$ строит успешный вывод, то доказательство завершено.

3. Если тактика $tac2$ не построила успешный вывод для пути из одного состояния, то для доказательства цели в состоянии доказательства применяется тактика $loop1$ с аргументами $0, n, \gamma, \delta, \delta, tac1, tac2$.

4.5.6. Главная тактика applicator

Главная тактика *applicator* предназначена для автоматического построения доказательства выполнимости формулы произвольного вида на модели.

```
lazymatch f with
| fAX f_1 =>
    let tac init_ :=
        let tac_1 := (applicator n f_1) in
        solve_fAX init_ tac_1
    in
    tac
| fOr f_1 f_2 =>
    let tac init_ :=
        let tac_1 := (applicator n f_1) in
        let tac_2 := (applicator n f_2) in
```

```
        solve_fOr init_ tac_1 tac_2
    in
    tac
| fAnd f_1 f_2 =>
    let tac init_ :=
        let tac_1 := (applicator n f_1) in
        let tac_2 := (applicator n f_2) in
        solve_fAnd init_ tac_1 tac_2
    in
    tac
| fAU f_1 f_2 =>
    let tac init_ :=
        let tac_1 := (applicator n f_1) in
        let tac_2 := (applicator n f_2) in
        solve_fAU n init_ tac_1 tac_2
    in
    tac
| fV k => solve_fV
| p => fail
end
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ f\ \beta)), \theta), init\_/\alpha \in \theta,$$

где f терм типа $form$. Параметрами алгоритма являются терм типа nat (количество состояний в модели), терм f типа $form$. Тогда выполняются следующие действия:

1. Терм f формулы, для которой применяется тактика, сопоставляется одним из его конструкторов.

2. В зависимости от конструктора $fAX, fOr, fAnd, fAU, fV$ применяется тактика $solve\_fAX, solve\_fOr, solve\_fAnd, solve\_fAU, solve\_fV$, параметризованная именем гипотезы $init\_$. Тактики, которые являются параметрами, для доказательства подформулы $\hat{f}$ формулы f применяются также согласно тактике $applicator\ n\ \hat{f}$.

3. Таким образом тактика *applicator*, строит тактику согласно структуре формулы (как одна формула вложена в другую)

Тактика *applicator* может применяться не для всех термов типа $form$, а только для термов, которые построены с помощью конструкторов $fAX, fOr, fAnd, fAU, fV$. Построение соответствующих вспомогательных тактик для доказательства формул оказалось достаточно трудозатратныйм, поэтому на данный момент вспомогательные тактики для доказательства формул другого вида не построены.

4.5.7. Главная тактика

Главная тактика *solver*, предназначенная для автоматического доказательства суждения о типе, записанного в формальной постановке задачи в разделе 4.1, устроена следующим образом.

```
lazymatch goal with
| satisfies model f st =>
    let tac := applicator n f in
    tac init_l
| p => fail
end.
```

Эта тактика отвечает следующему алгоритму построения доказательства. Пусть на вход алгоритму подан набор состояний доказательства $g = (g_1)$, где

$$g_1 = ((\Gamma, (\alpha : \beta = v) \vdash (satisfies\ model\ f\ \beta)), \theta), init\_/\alpha \in \theta.$$

Параметром алгоритма является терм n типа nat (количество конструкторов типа *state model*) и имя гипотезы α типа $\beta = v$.

Тогда выполняются следующие действия:

1. Текущая цель в состоянии доказательства сопоставляется с термом $satisfies$ и его аргументами.

2. Если сопоставление успешно, то есть в доказываемом терме действительно утверждается выполнимость некоторой формулы на некоторой модели, то вызывается центральная вспомогательная тактика, строящая доказательство для формулы и модели, извлечённых при сопоставлении.

3. Иначе доказательство завершается неуспехом (ошибкой).

4.6. Корректность главной тактики

Интерактивное средство *Coq* доказательства теорем можно условно разделить на две состовляющие: небольшое ядро, предоставляющее средства проверки типа заданного терма, и обширный спектр модулей, надстраивающихся над этим ядром и предоставляющих конкретные индуктивные типы, средства работы с этими типами, тактики и т.п.

Ядро имеет небольшой размер и относительно просто реализует относительно небольшой набор понятий, и отладке ядра уделялось много внимания в течение нескольких десятилетий. Поэтому считается, что наличие ошибок в ядре очень маловероятно, и принято доверять результатам проверки ядра по крайней мере настолько же, как если бы соответствующие выкладки были оформлены общепринятыми способами на естественном языке квалифицированным математиком в научной публикации, и даже больше[20].

Согласно изоморфизму Карри–Говарда (см., раздел 2.2.5), доказательство утверждения A в средстве *Coq* представляет собой терм, имеющий тип A, и построение этого доказательства представляет собой вывод суждения о типе с подходящим термом-доказательством средствами ядра[15]. Поэтому все доказательства в *Coq* принято считать корректными - настолько же, насколько и ядро *Coq*, а значит, просто корректными. В частности, это означает, что и тактика построения доказательства выполнимости формулы *CTL* на модели Крипке, предложенная в разделе 4.5.7, считается корректной: если доказательство, построенное при помощи средств *Coq* (базовых тактик и языка *Ltac*) успешно завершено, то оно считается корректным. Остаётся только вопрос, для каких именно теорем доказательство будет успешно построено - но это вопрос полноты, обсуждающийся далее в разделе 4.7.

4.7. Полнота главной тактики

Теорема 1. *Пусть выполнены следующие условия:*

1. *model - произвольный терм типа sts, соответствующий модели Крипке M согласно изложенному в разделе 2.1 и построенный согласно ограничениям раздела 4.4.*

2. *f - произвольный терм типа form, не содержащий конструкторы fF, fImp, fAR (то есть содержащий только конструкторы fV, fAnd, fOr, fAX, fAU) и*

соответствующий формуле CTL φ согласно изложенному в разделе 2.1.

3. $M \models \varphi$

4. $g = (g_1)$ - *набор состояний доказательства, имеющий следующий вид*

$$g_1 = ((\Gamma_{MC}, (\alpha : \beta = t) \vdash satisifies\ model\ f\ \beta), \theta_i),$$

где t - конструктор, отвечающий начальному состоянию M, Γ_{MC} – контекст описанный в разделе 4.1.

5. n - *терм типа nat, отвечающий числу конструкторов типа state model.*

Тогда тактика solver с параметрами n и α строит успешный вывод для набора g.

Доказательство. *Доказательство будем проводить с помощью индукции по формуле f.*

База индукции: *$f = fV\ k$, где k – терм типа nat. Это означает, что согласно построенным ограничениям после применения правил редукции цель состояния g будет иметь следующий вид*

$$(t = constr_1 \to A_1) \wedge ... \wedge (t = constr_j \to (k = j_1 \vee ... \vee k = k \vee ... \vee k = j_m) \wedge ... \wedge (t = constr_n \to A_n).$$

При выполнении тактики applicator $(fV\ k)$ выбирается и выполняется тактика $solve\_fV$ с параметром α. Так как $M \models \varphi$, то для каждой скобки: если t совпадает с $constr_j$, то справа от импликации в дизъюнкции есть тип $k = k$, а значит, по устройству тактики $(repeat((left; reflexivity) + (right; reflexivity) + right))$, она успешно завершит доказательство этой цели, иначе слева от ипликации можно построить равенство $t = constr_j$ двух разных конструкторов, а значит, по устройству тактики $(rewriteinit_i npre; discriminate)$, она успешно завершит доказательство этой цели.

Индуктивный переход.

Согласно условию, f имеет вид $(AX\ f_1)$, $(f_1 \vee f_2), (f_1 \wedge f_2)$ или $(AU\ f_1\ f_2)$. Рассмотрим каждый из этих случаев.

Случай 1. *$f = fAX\ f_1$. Это означает, что согласно построенным ограничениям состояния доказательства будут иметь следующий вид*

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ (fAX\ f_1)\ \beta), \theta_i).$$

При выполнении тактики applicator n $(fAX\ f_1)$ *выбирается и выполняется тактика solve_fAX с параметром* α. *Так как* $M \models \varphi$, *то согласно построению тактики solve_fAX (см. пункт 5 пояснений в разделе 4.5.4) преобразует состояние доказательство в состояние вида*

$$((\Gamma_i, (\alpha : \gamma = \hat{v_i}), (\gamma : state\ model), (\zeta : (trans\ model)\ \beta\ \gamma)$$

$$\vdash (satisfies\ model\ f_1\ \gamma)), \theta_i).$$

А по индуктивному предположению, тактика applicator n f_1 *строит успешный вывод для целей вида*

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_1\ \gamma), \theta_i).$$

Таким образом, applicator n f_1 *с аргументом* α *по предположению индукции построит успешный вывод для состояния доказательства, которое было получено применением тактики solve_fAX.*

Случай 2. $f = (fOr\ f_1\ f_2)$. *Это означает, что согласно построенным ограничениям состояния доказательства будут иметь следующий вид*

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ (fOr\ f_1\ f_2)\ \beta), \theta_i).$$

При выполнении тактики applicator n $(fOr\ f_1\ f_2)$ *выбирается и выполняется тактика solve_fOr с параметром* α. *Так как* $M \models \varphi$, *то согласно построению тактики solve_fOr (см. пункт 2 пояснений в разделе 4.5.2) преобразует состояние доказательство в состояние вида*

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ f_1\ \beta), \theta_i).$$

А по индуктивному предположению, тактика applicator n f_1 *строит успешный вывод для состояния доказательства вида*

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_1\ \gamma), \theta_i).$$

И если формула, отвечающая терму f_1, *выполняется на модели, то тактика applicator n* f_1 *с аргументом* α *по предположению индукции построит успешный вывод. Иначе формула, отвечающая* f_1, *не выполняется на модели, и по построению тактики*

solve_fOr (см. пункт 2 пояснений в разделе 4.5.2) эта тактика преобразует состояние в состояние вида

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ f_2\ \beta), \theta_i).$$

А по индуктивному предположению тактика applicator n f_2 строит успешный вывод для состояния доказательства

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_2\ \gamma), \theta_i).$$

Таким образом, тактика applicator n f_2 с параметром α строит успешный вывод, и соответственно тактика solve_fOr тоже.

Случай 3. *$f = (fAnd\ f_1\ f_2)$. Это означает, что согласно построенным ограничениям состояния доказательства будут иметь следующий вид*

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ (fAnd\ f_1\ f_2)\ \beta), \theta_i).$$

При выполнении тактики applicator n $(fAnd\ f_1\ f_2)$ выбирается и выполняется тактика solve_fAnd с параметром α. Так как $M \models \varphi$, то согласно построению тактики solve_fAnd (см. пункт 2 пояснений в разделе 4.5.3) преобразует состояние доказательство в состояния вида

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ f_1\ \beta), \theta_i),$$

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ f_2\ \beta), \theta_i).$$

А по индуктивному предположению, тактики applicator n f_1 и applicator n f_2 строят успешный вывод соответственно для состояний доказательства вида

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_1\ \gamma), \theta_i),$$

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_2\ \gamma), \theta_i).$$

Таким образом, по предположению индукции о том, что тактики applicator n f_1 и applicator n f_2 строят успешный вывод для подформул, отвечающих соответственно термам f_1 и f_2, тактика solve_fAnd строит успешный вывод.

Случай 4. *$f = (fAU\ f_1\ f_2)$. Это означает, что согласно построенным ограничениям состояния доказательства будут иметь следующий вид*

$$((\Gamma_i, (\alpha : \beta = t) \vdash satisifies\ model\ (fAU\ f_1\ f_2)\ \beta), \theta_i).$$

При выполнении тактики applicator n ($fAU\ f_1\ f_2$) выбирается и выполняется тактика $solve\_fAU$ с параметром α.

А по индуктивному предположению, тактики applicator n f_1 и applicator n f_2 строят успешный вывод соответственно для состояний доказательства вида

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_1\ \gamma), \theta_i),$$

$$g_i = ((\Gamma_i, (\hat{\alpha} : \gamma = \hat{t}) \vdash satisifies\ model\ f_2\ \gamma), \theta_i).$$

Так как $M \models \varphi$, то согласно построению тактики $solve\_fAU$ (см. пункт 1 пояснений в разделе 4.5.5) преобразует состояние доказательство в состояния вида

$$((\Gamma_i, (\beta : pi\ 0 = t), \vdash$$

$$p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)pi), \theta_i)$$

Рассмотрим путь, состоящий из одного состояния $pi\ 0 = t$. И согласно построению тактики $solve\_fAU$ (см. пункт 2 пояснений в разделе 4.5.5) состояние доказательства преобразуется в состояния доказательства с целями вида:

$$\Pi(m : nat), m < 0 \rightarrow\ satisfies\ model\ f_1\ (pi\ m),$$

$$satisfies\ model\ f_2\ (pi\ 0)$$

Для первого состояния вывод строится с помощью противоречия $m < 0$ и тактики lia. Для второго состояния по предположению индукции вывод строит тактика applicator n f_2 с гипотезой β в качестве параметра.

Если для этого пути тактики не строят успешный вывод. Рассмотрим путь, состоящий из двух состояний $pi\ 0 = t, pi\ 1 = t_1$. Выполняется тактика loop1, и согласно построению тактики loop1 (см. пункт 4 пояснений в разделе loop1) состояние доказательства будет иметь вид

$$((\Gamma_i, (\beta : pi\ 1 = t_1), (\gamma : pi\ 1 = t_1 \wedge pi\ 0 = t) \vdash$$

$$p\_until(state\ model)(satisfies\ model\ f_1)(satisfies\ model\ f_2)pi), \theta_i).$$

И выполняется тактика $proof\_ex\_sat$, согласно построению тактики $proof\_ex\_sat$ (см. пункт 2.1 и 3.1 пояснений в разделе $proof\_ex\_sat$) цели состояний доказательства будут иметь следующий вид

$$\Pi(m : nat), m < 1 \rightarrow\ satisfies\ model\ f_1\ (pi\ m),$$

$satisfies\ model\ f_2\ (pi\ 1)$

Для первого состояния применяется подвыражение loop (см. пункт 2.2.3 пояснений в разделе $proof\_ex\_sat$), используя тактику applicator n f_1, которая строит успешный вывод по предположению индукции. Для второго состояния по предположению индукции вывод строит тактика applicator n f_2 с гипотезой β в качестве аргумента.

Если для этого пути тактики не строят успешный вывод, то рассматривается путь с еще одним состоянием до тех пор, пока не построится успешный вывод.

*Тактика loop*1 *действует до тех пор пока в состояний в пути не становится равным n (числу состояний/конструкторов в модели Крипке). Эта тактика использует тот факт, что для проверки выполнимости формулы AU на модели Крипке достаточно проверить все пути, которые имеют количество состояний меньше либо равно чем количество состояний в модели Крипке[12].*

Таким образом, по предположению индукции о том, что тактики applicator n f_1 и applicator n f_2 строят успешный вывод для подформул, отвечающих соответственно термам f_1 и f_2, тактика $solve\_fAU$ строит успешный вывод.

5. Полученные результаты

В магистерской диссертации получены следующие результаты:

1. Формально поставлена задача проверки модели в терминах исчисления индуктивных конструкций (раздел 4.1).

2. Предложены формальные синтаксис и семантика фрагмента языка *Ltac* и базовых тактик, которые используются для решения задачи проверки модели в терминах исчисления индуктивных конструкций (разделы 4.2, 4.3).

3. Разработан алгоритм (раздел 4.5.7), записанный в терминах формализованных элементов языка *Ltac*, который доказывает выполнимость формулы *CTL*, построенной над операциями p, AX, $\vee$, $\wedge$, AU, на модели Крипке, представленной согласно разделу 4.4.

4. Обоснована полнота предложенного алгоритма (раздел 4.7) и отмечено, что обоснование корректности не требуется (раздел 4.6).

Список литературы

[1] Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002.

[2] Blazy S., Leroy X. Formal verification of a memory model for C-like imperative languages. In International Conference on Formal Engineering Methods (ICFEM 2005), Springer, Vol. 3785, 2005, p. 280-299.

[3] Pierce B. C. Software Foundations Volume 5: Verifiable C. Electronic textbook. [Электронный ресурс]. - Электрон. дан. - URL: https://softwarefoundations.cis.upenn.edu/vc-current/index.html. (дата обращения 14.02.2023)

[4] The Mathematical Components Library [Электронный ресурс]. - Электрон. дан. - URL: https://github.com/math-comp/math-comp. (дата обращения 21.05.2022)

[5] Logic for Programming and Automated Reasoning. LPAR 2000. Lecture Notes in Artificial Intelligence, vol 1955. Springer, Berlin, Heidelberg. P. 85-95.

[6] Tsai, MH., Wang, BY. Formalization of CTL* in Calculus of Inductive Constructions. Secure Software and Related Issues. ASIAN 2006. Lecture Notes in Computer Science, Springer, Vol. 4435, 2007, p. 316-330.

[7] Doczkal, C., Smolka, G. (2016). Completeness and Decidability Results for CTL in Constructive Type Theory. Journal of Automated Reasoning, Vol. 56, 2016, p. 343–365.

[8] Doczkal C. A Machine-Checked Constructive Metatheory of Computation Tree Logic. PhD Thesis. Saarland University, 2015.

[9] Coupet-Grimal S. An Axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions. Journal of Logic and Computation, Vol. 13, 2003, p. 801–813.

[10] Besson F. Fast Reflexive Arithmetic Tactics the Linear Case and Beyond. Types for Proofs and Programs, Springer, Berlin, Heidelberg, 2007, p. 48-62.

[11] Ltac. Documentation. [Электронный ресурс]. - Электрон. дан. - URL: https://coq.inria.fr/refman/proof-engine/ltac.html. (дата обращения 14.12.2022)

[12] Карпов Ю. Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010.

[13] Paulin-Mohring C. Introduction to the Calculus of Inductive Constructions. All about Proofs, Proofs for All, Vol. 55, College Publications, 2015, Studies in Logic (Mathematical logic and foundations).

[14] Coquand T., Huet G. The calculus of constructions. Information and computation, Vol. 76, 1988, p. 95-120.

[15] Howard W.A. The formulae-as-types notion of constructions. To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 1980, p. 479–490.

[16] Coq. Documentation. [Электронный ресурс]. - Электрон. дан. - URL: https://coq.inria.fr/refman/proofs/writing-proofs/proof-mode.html. (дата обращения 14.02.2023)

[17] Coq. Documentation. [Электронный ресурс]. - Электрон. дан. - URL: https://coq.inria.fr. (дата обращения 10.02.2023)

[18] Completeness and Decidability of Modal Logic Calculi [Электронный ресурс]. - Электрон. дан. - URL: https://github.com/coq-community/comp-dec-modal. (дата обращения 21.05.2022)

[19] Jedynak W. Operational Semantics of Ltac. Master Thesis. Uniwersytet Wroclawski, 2013.

[20] Sozeau M., Boulier S., Forster Y., Tabareau N., Winterhalter T. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. Proc. ACM Program. Lang., Vol. 4, 2020, Article 8, p. 1-28.