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

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, которые имеют высокую степень доверия, и оно же проверяет правильность построенного доказательства. Правильность работы самого средства *Coq* обеспечивается засчет того, 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 поэтапного 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][7][8], посвященные анализу темпоральных логик в *Coq*. В работе [8] 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 работе [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.

Целью магистерской диссертации были формальная постановка задачи проверки модели терминах *Coq*, построение формальных синтаксиса и семантики языка *Ltac* и базовых тактик, основываясь на их документации [9] и на работе [5], содержащие описания, близкие к формальным, но не вполне строгие, разработатка алгоритма, записанного в терминах формализованных элементов языка *Ltac* и решающего рассматриваемую задачу проверки модели, обоснование корректности и полноты построенного алгоритма. В результате, была формализована задача проверки модели в терминах *Coq*, была построена основная часть формального синтаксиса и семантики языка *Ltac*, необходимого для описания алгоритма, был разработан алгоритм, записанный в терминах формализованных элементов языка *Ltac* и решающий рассматриваемую задачу проверки модели, были обоснованы корректность и полнота разработанного алгоритма.

2. Основные понятия

2.1. Логика ветвящегося времени (CTL)

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 [10] 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 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:

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

2.2.1. 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 [11], однако для дальнейшего описания эти ограничения не потребуются. Далее будут использоваться индуктивные типы, которые выполняют эти ограничения и проверены средством *Coq*.

Будем записывать, что $x \in y$, 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 $O,\ (S\ O),\ (S\ (S\ O))...$

2.2.2. 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 $I\ parameters$.

2.2.3. 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:

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

2.2.4. Исчисление индуктивных конструкций (CIC)

Исчисление индуктивных конструкций (CIC) – это расширение исчисления конструкций[12], в котором есть возможность определять индуктивный тип, строить термы, используя дополнительно выражения $match$ и $fixpoint$.

2.2.5. Соответствие Карри–Говарда

Соответствие Карри-Говарда [13] применительно к исчислению индуктивных конструкций [14] - это соответствие между элементами исчисления индуктивных

конструкций и исчислений, предназначенных для конструктивного доказательства высказываний на языке логики предикатов. Это соответствие устроено следующим образом.

- Квантору Π 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 предполагается 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 полноту описанного алгоритма

5. Обосновать корректность описанного 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. В постановке задачи использовались индуктивные типы, лежащие в основе программного кода из стандартной библиотеки[15] *Coq* и из работы[7][16]. Построенное математическое описание этих индуктивных типов представлено ниже. Удобное программное представление модели Крипке было перестроено в индуктивный тип, в который был добавлен параметр *init* для обозначения множества начальных состояний для соответствия с постановкой задачи из раздела 2.1. После этого потребовалось добавить термы, соответствующие отображениям, которые на вход принимают терм модели Крипке и возвращают один из параметров его конструктора.

Программный код на языке *Coq*, который соответствует формализации постановки задачи приведен в приложении.

4.1.1. Вспомогательные индуктивные типы

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:

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

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

Формализация модели Крипке в терминах *CIC*

Индуктивный тип sts, соответствующий моделям Крипке, устроен так:

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

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 $Prop$. Множество термов t, для который 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

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

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 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 индуктивного типа $form$ соответствуют элементам укороченного синтаксиса формул CTL:

- Конструктор fF соответствует $false$.
- Конструктор fV соответствует p.
- Конструктор $fImp$ соответствует $\to$.
- Конструктор $fAnd$ соответствует $\wedge$.
- Конструктор fOr соответствует $\vee$.
- Конструктор fAX соответствует AX.
- Конструктор fAU соответствует AU.
- Конструктор fAR соответствует 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 подтерма $match$ 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

$$\Gamma \vdash t : Sat,$$

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

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 [17], посвященной полному описанию синтаксиса и семантики, используемый математический аппарат является затруднительным для использования в данной работе. Документация же по языку *Ltac* [15] дает лишь нестрогое описание синтаксиса и семантики.

В данном разделе представлено описание части синтаксиса и семантики языка *Ltac*.

Запись $\Gamma \vdash A$ является целью. Цель $\Gamma \vdash A$ означает, что есть терм t типа A, для которого верно $\Gamma \vdash t : A$. Состояние доказательства – это пара, состоящая из цели и оценки параметров. Состояние вычисления — это пара (l, s), где l — состояние управления, а s — состояние данных.

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
```

```
| 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` $\in$ $term$, L - множество всех *Ltac*-выражений, `parameters`- набор параметров, где каждый параметр из множества $Tactic \bigcup Name \bigcup term \bigcup L$ и `ltac_expr` - *Ltac*-выражение, построенное на языке *Ltac*. Таким образом, состояние доказательства – это пара $(\Gamma \vdash A, \theta)$, где первый элемент – это цель, а второй – оценка имен $\theta = [a_0/l_0, ..., a_n/l_n]$. Каждое имя $a_0, ...a_n \in Name$ в оценке параметров отлично от других имен в этой оценке, и имен базовых тактик. Значение оценки имени – это имя, терм или *Ltac*-выражение. Будем обозначать тот факт, что существует оценка с именем a_i значением l_i, 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 $(l_1|g_1) \Rightarrow (idtac|g_2)$, то $(l_1; l_2|g_1) \rightarrow (l_2|g_2)$.

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

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

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 $(repeatl_1|g_1) \rightarrow (repeatl_1|g_2)$.

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 $p_1, ..., p_n, p_i \in term \bigcup Name \bigcup Tactic$ – 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.

Будем говорить, что *Ltac*-выражение l строит успешный вывод, если вычисление $solve[l]$ не приводит к ошибке ($\perp$).

4.3. Синтаксис и семантика базовый 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:

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

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

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

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:

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

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

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

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, 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, 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, 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 eq, le и тому подобных. 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 $h_i = (\Gamma_i \vdash B_i, \theta_i)$, где в B_i применены правила редукции.

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

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

где $h = (h_1, ..., h_n)$, где $h_i = ((\Gamma, (\alpha : A_i) \vdash C_i), \theta_i)$, где в A_i применены правила редукции.

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

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

Терм индуктивного типа sts, изложенный в параграфе 4.1.2 позволяет описать термы соответствующим моделям Крипке с неограниченным числом состояний, поэтому далее будем рассматривать для создания терма типа sts индуктивные типы A, которые имеют конечное число термов и имеют n конструкторов вида $constr_1 : A$,...,$constr_n : A$.

Введем ограничение на представление отношения преходов для рассматриваемых моделей Крипке.

Введем вспомогательные индуктивные типы $prod$ и $list$, которые соответственно описывают пары и списки термов:

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

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

Для задания отношения перехода представим модель Крипке как ориентированный граф $G = (V, E)$, в V–множество термов состояний, E–множество пар термов (t_1, t_2), которые находятся в отношении перехода. Одним из представлений ориентированных графов является список смежности. Список смежности – это список пар, где первый элемент вершина, из которой исходит дуга, и где второй элемент, список вершин куда входит дуга. Введем ограничение, что список смежности должен содержать каждый конструктор ровно один раз как первый элемент пары. Пусть задан индуктивный тип A, термы которого описывают состояния модели Крипке, пусть задан терм t типа $list(prodA(listA))$, описывающий список смежности графа модели Крипке.

Введем вспомогательные термы для преобразования списка смежности в терм, необходимого для задания модели Крипке.

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

$(match\ states\_in\ with$

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

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

$|\ nil \Rightarrow False$

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

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

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

$(match\ list\_conn\ with$

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

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

$|\ nil \Rightarrow False$

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

Таким образом, для описания отношения перехода при создании терма модели Крипке будем использовать следующий терм

$$(\lambda(s_1 : A), \lambda(s_2 : A), make\_prop\ s_1\ s_2\ t) : A \to A \to Prop.$$

То есть после подстановки аргументов s_1, s_2 и применения правил редукции терм примет следующий вид

$(s_1 = constr_1 \to (s_2 = constr_{11} \vee ... \vee s_2 = constr_{1m})) \wedge ... \wedge (s_1 = constr_n \to (s_2 = constr_{n1} \vee ... \vee s_2 = const$

Этот тип соответствует дугам $(constr_i, constr_{ij}), i = 1, ..., n, j = i_1, ..., i_m$.

Для задания отношения между состоянием и событиями будем использовать схожий со списком смежности список пар, где первый элемент вершина, а второй список из термов nat. Пусть задан терм l типа $list(prod\ A\ (list\ nat))$ описывающий этот список.

Таким образом, для описания разметки для каждого состояния модели Крипке будем использовать следующий терм

$$(\lambda(a : nat), \lambda(b : A), make\_prop\ b\ a\ l) : nat \to A \to Prop.$$

То есть после подстановки аргументов a, b и применения правил редукции терм примет следующий вид

$$(b = constr_1 \to (a = k_{11} \vee ... \vee a = k_{1m})) \wedge ... \wedge (b = constr_n \to (a = k_{n1} \vee ... \vee a = k_{nm})).$$

Этот тип соответствует разметке для состояния $constr_i$ переменными $x_{k_{ij}}), i = 1, ..., n, j = i_1, ..., i_m$.

Множество начальных состояний ограничим одним состоянием v, и терм описывающий утверждение, что терм v является начальным состоянием – это

$$(\lambda(s : A),\ eq\ s\ v) : A \to Prop.$$

Терм, соответствующий доказательству тотальности модели, будем брать произвольным.

4.5. Алгоритмы

Пусть заданы в контексте $model := t : sts$, где – t терм типа sts, построенный согласно описанным выше ограничениям.

4.5.1. solve_fV

Терм алгоритма, для доказательства утверждения о выполнимости формулы вида $fV\ n$, где n–терм типа nat, имеет следующий вид.

```
compute;
repeat split;
let pre in
intro pre;
( rewrite init_ in pre; discriminate )
+
( repeat ( (left; reflexivity) + (right; reflexivity) + right) )
```

Алгоритм принимает на вход имя гипотезы α типа $\beta = v$.

Алгоритм работает следующим образом:

1. Алгоритм доказывает утверждение для состояний доказательства $g = (g_1, ..., g_n)$, где

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

2. После применения всех правил редукции *compute*, согласно описанным выше ограничениям модели цель в состояниях доказательства будет иметь вид

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

3. Далее применяется комбинация *repeat split*, которая «разбивает» состояния доказательства на состояния вида

$$((\Gamma, (\alpha : \beta = v) \vdash (\beta = constr_i) \to (n = n_{i1} \vee ... \vee n = n_{im})), \theta_{ij}).$$

   После применения *intro pre* соответственно:

$$((\Gamma, (\alpha : \beta = v)(\gamma : \beta = constr_i) \vdash (n = n_{i1} \vee ... \vee n = n_{im})), \theta_{ij}).$$

4. Далее возможно две альтернативы:

   4.1. Если термы v и $constr_i$ не совпадают, значит можно построить гипотезу о равенстве двух неравных термах, что позволяет доказать любую цель. Выражение $rewrite\ init\_\ in\ pre;\ discriminate$ делает это.

   4.2. Если термы v и *constr* совпадают, то перебираются элементы дизъюнкции до тех пор, пока не найдется тот, который слева и справа от равенства записаны одинаковые термы типа *nat*, и доказывается это равенство. Выражение $repeat((left; reflexivity) + (right; reflexivity) + right)$ делает это.

4.5.2. solve_fAnd и solve_fOr

Терм алгоритма для доказательства утверждения о выполнимости формулы вида $(fOr\ f_1\ f_2)$, где f_1, f_2– подформулы, представлен ниже.

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

Параметром алгоритма является имя гипотезы типа $\beta = v$, и две тактики для доказательства подформул.

Алгоритм работает следующим образом:

1. Алгоритм доказывает утверждение для состояния доказательства $g = (g_1, ..., g_n)$, где

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

   И согласно определению $satisfies$ состояние доказательства после применения правил редукции может иметь вид

$$g_i = ((G_i, (\alpha : \beta = v) \vdash (satisfies\ model\ (f_1)\ \beta) \vee (satisfies\ model\ (f_2)\ \beta)), \theta_i).$$

2. Далее применяется подтактика $tac1$ к левой части дизъюнкции, если подтактика $tac1$ строит успешный вывод, то конец, иначе применяется подтактика $tac2$ к правой части дизъюнкции. Если она не строит успешный вывод, то $\perp$.

Терм алгоритма, для доказательства утверждения о выполнимости формулы вида $(fAnd\ f_1\ f_2)$, где f_1, f_2– подформулы.

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

Параметром алгоритма является имя гипотезы типа $\beta = v$, и две тактики для доказательства подформул.

Алгоритм работает следующим образом:

1. Алгоритм доказывает утверждение для состояния доказательства $g = (g_1, ..., g_n)$, где

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

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

$$g_i = ((G_i, (\alpha : \beta = v) \vdash (satisfies\ model\ f_1\ \beta)), \theta_i),$$

$$g_i = ((G_i, (\alpha : \beta = v) \vdash (satisfies\ model\ f_2\ \beta)), \theta_i).$$

   И для каждой применяется соответствующая тактика $tac1$ или $tac2$.

4.5.3. solve_fAX

Терм алгоритма, для доказательства утверждения о выполнимости формулы вида $(fAX\ f)$, где f– подформула, представлен ниже.

```
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_ ]
```

Параметром алгоритма является имя гипотезы типа $\kappa = v$ и тактика для доказательства подформул.

Алгоритм работает следующим образом:

1. Алгоритм доказывает утверждение для состояний доказательства $g = (g_1, ..., g_n)$, где

$$g_i = ((G_i, (\alpha : \kappa = v) \vdash (satisfies\ model\ (fAX\ f)\ \kappa)), \theta_i), init\_/\alpha \in \theta_i.$$

2. Сначала применяются правила редукции к цели (тактика `compute`), затем предположения «переносятся в гипотезы» (тактика `intro`). На данном этапе состояние доказательства имеет вид:

$$((G_i, (\alpha : \kappa = v)(\gamma : state\ model)(\beta : (trans\ model)\ \kappa\ \gamma)$$

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

$$next\_state/\gamma, trans\_to\_next\_state/\beta \in \theta_i$$

3. И согласно ограничениям после применения правил редукции тип гипотезы β будет иметь следующий вид

   $$(\kappa = constr_1 \to (\gamma = v_{11} \vee ... \vee \gamma = v_{1m})) \wedge ... \wedge (\kappa = constr_n \to (\gamma = v_{n1} \vee ... \vee \gamma = v_{nm}))$$

4. Так как по условию в каждой скобке слева от имликации присутствует каждый конструктор, и если v является конструктором $constr_i$, то подвыражение $loop...$ преобразует тип гипотезы α в $(\gamma = v_{i1} \vee ... \vee \gamma = v_{im})$

5. Далее с помощью выражения $repeat(...)$ состояние доказательства «разбиваются» на im состояний, то есть состояние доказательства

   $$((G_i, (\alpha : (\gamma = v_{i1} \vee ... \vee \gamma = v_{im}))(\gamma : state\ model)(\beta : (trans\ model)\ \kappa\ \gamma)$$
   $$\vdash (satisfies\ model\ f\ \alpha)), \theta_i),$$

   заменяется на im состояний вида

   $$((G_i, (\alpha : (\gamma = v_{ij})(\gamma : state\ model)(\beta : (trans\ model)\ \kappa\ \alpha)$$
   $$\vdash (satisfies\ model\ f\ \gamma)), \theta_i), j = 1, ..., im$$

6. Далее каждая полученная цель доказывается тактикой $tac1$ полученной в качестве аргумента.

4.5.4. solve_fAU

Рассмотрим вспомогательные тактики.

unsplit Это тактика, которая из двух гипотез создает третью, которая является конъюнкцией типов первых двух гипотез.

```
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 Это тактика, которая на основе гипотезы о текущем состоянии в пути создает гипотезу о состояниях в следующий момент времени. Тактика принимает на вход: i- терм типа nat, is_path_pi– гипотезу типа $\Pi(n : nat), (trans\ model)(pi\ n)(pi(S\ n))$, $first\_state$ гипотеза типа $pi\ i = v$, где v – это один из конструкторов типа $(state\ model)$, где pi имеет тип $nat \to (state\ model)$.

```
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_i first_state;
repeat (
    lazymatch type of first_state with
    | or l r  =>
        destruct first_state as [first_state | first_state]
    | r =>
        idtac
    end
)
```

Тактика работает следующим образом:

1. Обозначим через A текущую цель в состоянии доказательства. Создается новая гипотеза $is\_path\_pi\_i$ (тактика $pose\ proof$), тип которой – это $(trans\ model)(pi\ i)(pi(S\ i))$. Согласно построенным ограничениям после применения редукции тип это гипотезы будет иметь следующий вид

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

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

2. Далее схожим образом, как в алгоритме $solve\_fAX$ строятся jm состояний доказательств для каждого состояния g_j

$$((G_j, (\alpha : pi(S\ i) = v_{jk}) \vdash A, k = 1, ..., jm.$$

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.

$$\hat{\Gamma} =$$

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

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

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

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

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

proof_ex_sat i seq tac1 tac2 Тактика $proof\_ex\_sat$ принимает на вход терм i типа nat, гипотезу seq типа $pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0$ и тактики $tac1$,$tac2$. Она применяется для построения вывода для состояний доказательств, цель которых имеет вид

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

где pi типа $nat \rightarrow (state\ model)$.

```
eexists i; compute;
[
    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
].
```

Тактика работает следующим образом:

1. Применяется тактика $eexists\ i$, и цель «разбивается» на два состояния, которые имеют следующие цели $\Pi(m : nat), m < i \rightarrow satisfies\ model\ f_1\ (pi\ m)$ и $satisfies\ model\ f_2\ (pi\ i)$

2.1 Рассмотрим первое состояние

$$((\Gamma_i, (\delta : pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0) \vdash \Pi(m : nat), m < i \rightarrow satisfies\ model\ f_1\ (pi\ m)), \theta_i),$$

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

которое после применения тактик *intro* и *destruct* примет вид

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

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

2.2 Далее проверяется, что является ли тип гипотезы $\hat{\delta}$ конъюнкцией.

2.2.1 Если тип есть *and*, то гипотеза $(\hat{\delta} : pi\ (i-1) = v_{i-1} \wedge ... \wedge pi\ 0 = v_0)$ снова «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 применяется гипотеза $usefull$ к гипотезе γ, и ее тип – это $S\ \kappa = i\ \vee\ \kappa < (i-1)$

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 $S\ \kappa = i$ преобразуется в $\kappa = i - 1$ и применяется тактика $tac1$ с аргументом $\hat{\delta_2}$ для цели этого состояния $satisfies\ model\ f_1\ (pi\ \kappa)$. Для второго состояния рекурсивно используется тактика *loop* с гипотезой $\hat{\delta_1}$ в качестве аргумента.

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

2.3.1 Далее применяется гипотеза $usefull$ к гипотезе γ, и ее тип – это $S\ \kappa = 1\ \vee\ \kappa < 0$.

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 $S\ \kappa = 1$ преобразуется в $\kappa = 0$ и применяется тактика $tac1$ с аргументом $\hat{\delta_2}$ для цели этого состояния $satisfies\ model\ f_1\ (pi\ \kappa)$. Для второго состояния применяется тактика *lia*, так как в гипотезах получили невозможное неравенство $\kappa < 0$, что позволяет доказать любую цель, включая текущую.

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

3.2. Применяется 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. Применяется тактика $tac2$ с гипотезой ϕ в качестве аргумента.

loop1 i n is_path_pi last_stop prev_acc tac1 tac2 Тактика $loop1$ принимает на вход терм i типа *nat*, терм n типа *nat* – это количество конструкторов индуктивного типа $(state\ model)$, гипотезу $last\_stop$ типа $pi\ i = v_i$, гипотезу $prev\_acc$ типа $pi\ i = v_i \wedge ... \wedge$

$pi\ 0 = v_0$ и тактики $tac1, tac2$. Она применяется для построения вывода для состояний доказательств, цель которых имеет вид

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

где pi типа $nat \to (state\ model)$.

```
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)
```

Тактика работает следующим образом:

1. Проверяется равенство $i = n$ тем, что может ли оно быть доказуемо. Если $i = n$ доказуем, то $\bot$.

2. Иначе создается новая гипотеза α в текущем состоянии доказательства:

$$((\Gamma_i, (\beta : pi\ i = v_i), (\alpha : pi\ i = v_i), (\gamma : pi\ (i-1) = v_{i-1} \wedge ... \wedge pi\ 0 = v_0) \vdash$$

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

3. Применяется тактика $next\_state\_gen$ и гипотеза α имеет тип $\alpha : pi\ (S\ i) = \hat{v_i}$

4. Далее создается новая гипотеза δ с помощью тактики $unsplit$ типа $pi\ (S\ i) = \hat{v_i} \wedge pi\ i = v_i \wedge ... \wedge pi\ 0 = v_0$

5. Затем, если удается доказать для данной «цепочки» δ, то вывод успешен.

6. Если не удается доказать для данной «цепочки» δ, рекурсивно вызывается $loop1$...

solve_fAU n init_l tac1 tac2 Терм алгоритма, для доказательства утверждения о выполнимости формулы вида $(fAU\ f_1\ f_2)$, где f_1, f_2 – подформулы, представлен ниже.

```
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)
```

Параметром алгоритма являются терм n типа nat (количество состояний в модели), имя гипотезы типа $\alpha = v$ и тактики для доказательства подформул. И строит доказательство для состояния доказательства следующего вида:

$$((\Gamma_i, (\beta : \alpha = v) \vdash satisfies\ model\ (fAU\ f_1\ f_2)\alpha), \theta_i)$$

Алгоритм работает следующим образом:

1. После применения тактик *intro* и *rewrite* состояние доказательства примет следующий вид

   $$((\Gamma_i, (\beta : \alpha = v), (\delta : \pi\ 0 = v), (\pi : nat \to (state\ model)), (\gamma : path(state\ model)(trans\ model)\pi) \vdash$$

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

2. Сначала применяется выражение $eexists\ 0; [...]$ 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 $tac2$, которая является аргументом этого алгоритма.

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

4.5.5. Тактика для автоматического построения выражения

Рассмотрим тактику *applicator*, которая строит тактику, необходимую для доказательства, рассматриваемой формулы. Тактика строит композицию описанных

выше тактик согласно формуле. Она принимает на вход терм типа $form$,терм типа nat, соответствующий количеству конструкторов индуктивного типа состояний, и возращает тактику для построения вывода.

```
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
```

Алгоритм работает следующим образом:

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

2. В зависимости от конструктора строится тактика параметризованная именем гипотезы $init\_$, в которой применяется соответствующая конструктору тактика, которой на вход подаются тактики для решения подформул, которые строятся рекурсивным способ для соответствующий подформул.

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

4.5.6. Тактика для автоматического применения

Для того, чтобы построить тактику, необходимую для решения задачи проверки модели, необходимо получить терм формулы, для которой необходимо построить доказательство. Для это используется следующее выражение *solver* с аргументами n – терм типа *nat* (количество конструкторов типа *state model*) и имя гипотезы $init\_l$ типа $st = v$.

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

Алгоритм работает следующим образом:

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

2. Одним из аргументов этого терма является формула-терм f, для которой строится проверка модели.

3. Далее f используется как аргумент *applicator* для построения тактики для построения вывода.

4.6. Корректность построенных алгоритмов

Интерактивное средство *Coq* доказательства теорем можно условно разделить на две состовляющие: ядро для проверки термов и инструменты для описания термов над этим ядром.

Ядро предназначено определения типа, который имеет конкретный терм. Ядро устроено относительно простым образом, так чтобы минимизировать вероятность ошибки, а исчисление конструкций (CIC), которое является теоретической основой ядра, надежно и непротиворечиво [12].

Одним из инструментов для описания термов являются тактики. Тактики — это небольшие программы, которые из построенного вывода строят терм. И тактика строит успешный вывод, если терм соответсвует тому типу, который требуется (это проверяется самим ядром), поэтому если тактика завершила успешно работу, то терм корректен.

Coq широко используется для проверки уже доказанных теорем в математике [4]. И проверяются теоремы так: если утверждение соответствующее исходной теореме можно доказать в *Coq*, то оно верно. Также можно сказать об утверждении о выполнимости формулы на модели: если алгоритм на тактиках построил успешный вывод, то верно утверждение о том, что формула выполняется на модели.

Таким образом, алгоритмы на тактиках, описанные в разделе 4.5, обладают свойством корректности.

4.7. Полнота построенных алгоритмов

Утверждение. Пусть выполнено следующие условия:

1. Терм $model$ типа sts, соответствующий модели Крипке, построен согласно ограничениям из раздела 4.4.

2. Терм f типа $form$ построен с помощью конструкторов $fV, fAX, fAnd, fOr, fAU$.

3. Состояние доказательства $g = (g_1, ..., g_l)$ имеет следующий вид:

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

   где t – один из конструкторов типа $state\ model$.

Тогда если для цели в текущем состоянии доказательства можно построить успешный вывод, то тактика $solver$ с аргументами n и α строит успешный вывод для текущего состояния доказательства, где n терм типа nat, соответствующий числу конструкторов типа $state\ model$.

Доказательство.

Доказательство будем проводить с помощью индукции по формуле f.

База индукции.

Пусть формула $f = fV\ k$, где k терм типа nat. Пусть формула, соответствующая этому терму, выполняется на модели. Это означает, что согласно построенным ограничениям после применения правил редукции цель текущего состояния доказательства будет иметь следующий вид

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

В таком случае, тактика $applicator$ сопоставит терм $fV\ k$ с тактикой $solve\_fV$. Тактика $solve\_fV$ с гипотезой α в качестве аргумента построит успешный вывод, так как для каждой скобки: если t совпадает с $constr_j$, то цель доказывается с помощью $reflexivity$, так как среди дизъюнкций есть тип $k = k$, иначе доказывается с помощью противоречия $t = constr_j, j \neq i$ и тактики $discriminate$.

Индуктивный переход.

Согласно условию f может быть термами вида $(AX\ f_1),\ (f_1 \vee f_2), (f_1 \wedge f_2), (AU\ f_1\ f_2)$. Рассмотрим каждый из этих случаев.

1. Пусть f– это терм $fAX\ f_1$ и тогда состояние доказательства имеет вид

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

В таком случае, тактика *applicator* сопоставит терм $fAX\ f_1$ с тактикой $solve\_fAX$. Тактика $solve\_fAX$ с гипотезой α согласно пункту 5 описания алгоритма 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* f_1 (обозначим как $tac1$) 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, $tac1$ с аргументом $\hat{\alpha}$ по предположению индукции построит успешный вывод для состояния доказательства, которое было получено применением тактики $solve\_fAX$.

2. Пусть $f = (fOr\ f_1\ f_2)$, то есть состояние доказательства имеет вид

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

В таком случае, тактика *applicator* сопоставит терм $fOr\ f_1\ f_2$ с тактикой $solve\_fOr$. По пункту 2 алгоритма $solve\_Or$ тактика преобразует состояние в состояние

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

По индуктивному предположению тактика *applicator* f_1 (обозначим как $tac1$) 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 для подформулы f_1 можно построить успешный вывод, то тактика $tac1$ с аргументом $\hat{\alpha}$ по предположению индукции сделает это. Иначе если тактика $tac1$ не построила успешный вывод, то согласно пункту 2 алгоритма $solve\_fOr$ 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* f_2 (обозначим как $tac2$) 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, $tac2$ строит успешный вывод, и соответственно тактика $solv\_fOr$ тоже.

3. Пусть $f = (fAnd\ f_1\ f_2)$, то есть состояние доказательства имеет вид

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

В таком случае, тактика *applicator* сопоставит терм $fAnd\ f_1\ f_2$ с тактикой $solve\_fAnd$. Согласно пункту 2 алгоритма $solve\_fAnd$ состояние доказательства преобразуется 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* f_1 (обозначим как $tac1$) и *applicator* f_2 (обозначим как $tac2$) 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, тактика $solve\_fAnd$ строит успешный вывод по предположению индукции о том, что тактики $tac1$ и $tac2$ строят успешный вывод для подформул.

4. Пусть $f = (fAU\ f_1\ f_2)$, то есть состояние доказательства имеет вид

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

В таком случае, тактика *applicator* сопоставит терм $fAU\ f_1\ f_2$ с тактикой $solve\_fAU$. А по индуктивному предположению тактики *applicator* f_1 (обозначим как $tac1$) и *applicator* f_2 (обозначим как $tac2$) строят успешный вывод соответственно для состояний доказательства вида

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

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

Рассмотрим алгоритм $solve\_fAU$. Согласно пункту 1 состояние доказательства будет иметь вид.

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

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

Рассмотрим путь, состоящий из одного состояния $pi\ 0 = t$. И согласно пункту 2 алгоритма $solve\_fAU$ 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 $tac2$ с гипотезой β в качестве аргумента.

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 4 пункту алгоритма $loop1$ 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 применяется тактика $proof\_ex\_sat$, согласно его пунктам 2.1 и 3.1 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 $loop$ из пункта 2.2.3 алгоритма $proof\_ex\_sat$, используя тактику $tac1$, которая строит успешный вывод по предположению индукции. Для второго состояния по предположению индукции вывод строит тактика $tac2$ 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 $loop1$ 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[10].

Таким образом, тактика $solve\_fAU$ строит успешный вывод по предположению индукции о том, что тактики $tac1$ и $tac2$ строят успешный вывод для подформул f_1 и f_2 соответственно.

Теорема доказана.

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. Предложены средства описания моделей Крипке и описаны ограничения построения терма, соответствующего модели Крипке (раздел 4.4).

4. Разработан алгоритм, записанный в терминах формализованных элементов языка *Ltac*, который доказывает выполнимость некоторого класса формул на заданной модели Крипке.

5. Обоснована полнота описанного алгоритма (раздел 4.7).

6. Обоснована корректность описанного алгоритма (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] Delahaye D. A Tactic Language for the System Coq. Logic Programming and Automated Reasoning, 2000.

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

[8] 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.

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

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

[11] 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).

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

[13] W.A. Howard. The formulae-as-types notion of constructions. In J.P. Seldin and J.R. Hindley, editors, to H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press, 1980.

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

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

[16] 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)

[17] Jedynak W. Operational Semantics of Ltac. Master Thesis. UNIWERSYTET WROCŁAWSKI, 2013.

[18] Sozeau, Matthieu, Boulier. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. Proc. ACM Program. Lang., Vol. 4, 2020, Article 8.