ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 59791— 2021

Информационные технологии

ОБЩАЯ ЛОГИКА (CL)

Основы семейства языков, основанных на логике

(ISO/IEC 24707:2018, NEQ)

Издание официальное

Москва
Российский институт стандартизации
2021

Предисловие

1 РАЗРАБОТАН Обществом с ограниченной ответственностью «Информационно-аналитический вычислительный центр» (ООО ИАВЦ)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 22 «Информационные технологии»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 25 октября 2021 г. № 1283-ст

4 Настоящий стандарт разработан с учетом основных нормативных положений международного стандарта ИСО/МЭК 24707:2018 «Информационные технологии. Общая логика (CL). Основы семейства языков, основанных на логике» (ISO/IEC 24707:2018 «Information technology — Common Logic (CL) — A framework for a family of logic-based languages», NEQ)

5 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.rst.gov.ru)

Содержание

Введение

Общая логика (CL) — это логическая структура, предназначенная для обмена и передачи информации. Структура позволяет использовать множество различных синтаксических форм (диалектов), которые с сохранением семантики можно перевести в общий синтаксис на основе XML.

У CL есть несколько новых функций, главными из которых являются синтаксис, допускающий конструкции «более высокого порядка», например использование квантора по классам или отношениям при условии сохранения теории модели первого порядка, и семантика, которая позволяет теориям описывать интенциональные сущности, например классы или свойства. Кроме того, в CL предусмотрены инструменты для обработки типов данных и именования, импорта и передачи содержимого через Интернет с помощью XML.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационные технологии

ОБЩАЯ ЛОГИКА (CL)

Основы семейства языков, основанных на логике

Information technology. Common Logic (CL). A framework for a family of logic-based languages

Дата введения — 2022—04—30

1 Область применения

В настоящем стандарте описано семейство логических языков, разработанных для представления информации и данных и обмена ими между разнородными компьютерными системами.

Следующие особенности имеют особую важность для построения настоящего стандарта:

- языки в семье имеют декларативную семантику. Благодаря этому можно понять значение выражений на этих языках, не используя интерпретатор для обработки таких выражений;
- языки в семье логически всесторонни — в самом общем виде они обеспечивают выражение произвольных логических высказываний первого порядка;
- языки можно перевести с сохранением семантики в общий синтаксис на основе XML, что облегчает обмен информацией между разнородными компьютерными системами.

В область применения настоящего стандарта входят следующие вопросы:

- представление информации в онтологиях и базах знаний;
- описание выражений, которые вводят в машины логического вывода или выводят из них;
- формальные интерпретации символов, выраженных с помощью языка.

Нижеследующее выходит за рамки настоящего стандарта:

- описание теории доказательств или правил вывода;
- описание переводчиков между обозначениями разнородных компьютерных систем;
- компьютерные операционные методы формирования отношений между символами в логической «вселенной дискурса» и людьми в «реальном мире».

В настоящем стандарте описаны синтаксис и семантика CL.

В настоящем стандарте также приведено определение абстрактного синтаксиса и связанной с ним теоретико-модельной семантики для специфичного расширения логики первого порядка. Цель состоит в том, чтобы представить содержимое любой системы, использующей логику первого порядка. Назначение настоящего стандарта — упростить обмен логической информацией первого порядка между системами.

Вопросы, касающиеся вычислимости, не рассматриваются в настоящем стандарте (включая эффективность, оптимизацию и т. д.).

2 Нормативные ссылки

В настоящем стандарте использована нормативная ссылка на следующий стандарт:

ГОСТ 33707—2016 (ISO/IEC 2382:2015) Информационные технологии. Словарь

Примечание — При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов в информационной системе общего пользования — на официальном сайте Федерального агентства по

Издание официальное

техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать версию этого стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

3 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями.

С целью использования в своих стандартах международные организации ИСО и МЭК поддерживают терминологические базы данных:

- платформа ИСО для онлайн-просмотра: доступна по адресу: http://www.iso.org/obp;
- платформа МЭК Электропедия (IEC Electropedia): доступна по адресу: http://www.electropedia.org/.

3.1 **аксиома** (axiom): Любое высказывание (3.15), утверждение или текст, которые считают истинными, из которых происходят другие высказывания, утверждения или тексты, либо которые сами вытекают из таких высказываний, утверждений или текстов.

Примечание — В вычислительной среде аксиома — это высказывание, которое никогда не требуется доказывать и которое само используют для доказательства других высказываний.

3.2 **концептуальный граф**; CG (conceptual graph; CG): Графическое или текстовое отображение символов, упорядоченных в соответствии со стилем теории концептуальных графов (3.3).

3.3 **теория концептуальных графов** (conceptual graph theory): Форма логики первого порядка, которая представляет собой квантор существования и конъюнкцию посредством утверждения логических конструктов, называемых понятиями и отношениями, которые организованы в абстрактный или визуально отображенный граф.

3.4 **CLIF** (CLIF): Текстовый формализм первого порядка с использованием основанной на LISP списочной записи.

Примечания

1 Это один из конкретных синтаксисов CL (описан в приложении А).

2 CLIF — это синтаксис на основе языка KIF, который применен в настоящем стандарте в иллюстративных целях. KIF (формат обмена знаниями), представленный Майклом Генесеретом [1], возник в рамках инициативы по обмену знаниями, спонсируемой Управлением перспективных исследовательских проектов Министерства обороны США. Название «KIF» не относится к этому синтаксису, чтобы отличить его от обычно используемых диалектов KIF. В настоящем стандарте отсутствуют какие-либо допущения относительно семантики KIF; в частности, не предполагается тождественность между CLIF и KIF.

3 Исторически аббревиатура CLIF означала Common Logic Interchange Format (Общий формат обмена данными логики). Тем не менее CLIF не имеет привилегированного положения среди диалектов CL (3.7), как может показаться из полного названия. Кроме того, рекомендуемым форматом обмена в сети Интернет является XCL.

3.5 **формат обмена концептуальными графами** (conceptual graph interchange format): Текстовая версия концептуальных графов (3.2).

Примечание — Иногда этот термин может иметь отношение к примеру символьной строки, соответствующей приложению Б и предназначенной для передачи точно такой же структуры и семантики, что и эквивалентный концептуальный граф.

3.6 **денотат** (denotation): Связь между именем или выражением и предметом, к которому оно относится.

Примечание — Также используют для обозначения именуемой вещи, то есть отсылает к имени или выражению.

3.7 **диалект общей логики** (CL dialect): Определенный экземпляр синтаксиса CL, который использует единую семантику CL или ее часть.

Примечание — Диалект может быть текстовым, графическим или иметь другую форму. По определению диалект также является совместимым языком (подробнее см. п. 7.1).

3.8 **расширяемый язык разметки eXtensible Common Logic Markup Language;** XCL (eXtensible Common Logic Markup Language; XCL): Синтаксис XML для CL.

3.9 **единица** (individual): <Интерпретации> одного элемента вселенной дискурса (3.17) интерпретации (3.12).

Примечание — Вселенная дискурса интерпретации — это совокупность всех его единиц.

3.10 **интернационализированный идентификатор ресурса;** IRI (internationalized resource identifier; IRI): Строка символов Unicode, предназначенная для использования в качестве синтаксиса идентификатора сети Интернет, способного поддерживать широкий спектр международных форм символов.

3.11 **надпись** (inscription): Линейная или графическая структура символов.

3.12 **интерпретация** (interpretation): Формальное описание значений имен в словаре диалекта CL (3.7) в терминах эталонной вселенной (3.18).

Примечания

1 Интерпретация CL, в свою очередь, определяет семантические значения всех сложных выражений диалекта, в частности значения истинности его высказываний (3.15), утверждений и текстов.

2 Более точное описание интерпретации приведено в 6.2.

3.13 **оператор** (operator): Выделенная синтаксическая роль, которую играет указанный компонент в пределах функционального термина (3.16).

Примечание — Денотат (3.6) функционального термина в интерпретации (3.12) определяется функциональным расширением денотата оператора вместе с денотатами аргументов.

3.14 **предикат** (predicate): Синтаксическая роль <Common Logic>, которую играет ровно один компонент в простом высказывании (3.15).

Примечание — Истинность простого высказывания в интерпретации (3.12) определяется относительным расширением денотата (3.6) сказуемого с денотатами аргументов.

3.15 **высказывание** (sentence): Выражение <Common Logic> в синтаксической форме традиционной логической формулы первого порядка.

Пример — Простое высказывание (см. 6.1.1.15), логическое высказывание (см. 6.1.1.14) или квантор (см. 6.1.1.13).

3.16 **термин** (term): Выражение <Common Logic>, обозначающее единицу (3.9), состоящее либо из имени, либо, рекурсивно, функционального термина, который относится к последовательности аргументов, которые сами по себе являются терминами.

Примечание — Языки традиционной логики первого порядка специально исключают кванторы предиката (3.14) и использование одного и того же имени в качестве предиката и аргумента в простых высказываниях (3.15), оба из которых разрешены (хотя и не требуются) в CL. Языки традиционной логики первого порядка подпадают под категорию пресуппозиционного диалекта CL с дискурсивной пресуппозицией «недискурса» для всех имен, используемых в качестве функциональных операторов (3.13) или предикатов, и «дискурса» для всех имен, используемых в качестве аргументов функциональных терминов и простых высказываний, а также для обязательных условий.

3.17 **вселенная дискурса, область дискурса** (universe of discource): Множество всех единиц (3.9) в интерпретации (3.12), то есть множество, по которому ранжируются кванторы.

Примечание — Должна быть подмножеством исходной вселенной (3.18) и может быть идентичной ему.

3.18 **исходная вселенная** (universe of reference): Набор всего необходимого для определения значений логических выражений в интерпретации (3.12).

Примечание — Должна быть надмножеством вселенной дискурса (3.17) и может быть идентичной ему.

4 Обозначения и сокращения

4.1 Обозначения

В настоящем стандарте используются следующие обозначения:

fun_I — сопоставление от UR_I до функций от UD_I^* до UD_I;

I — интерпретация в теоретико-модельном смысле;

int_I — сопоставление от имен в словаре V до UR_I; неформально является средством соотнесения имен в V с референтами в UR_I;

rel_I — сопоставление от UR_I до подмножеств UD_I^*;

seq_I — сопоставление от маркеров последовательности в V до UD_I^*;

λ — лексикон, который состоит из словаря, набора маркеров последовательности (Smark) и набора заголовков (Ttl);

V — словарь, который представляет собой набор имен;

Smark — набор маркеров последовательности;

Ttl — набор заголовков;

UD_I — вселенная дискурса; непустое множество единиц, которые описывает интерпретация I и по которым ранжируются кванторы;

UR_I — исходная вселенная, т. е. совокупность всех референтов имен в интерпретации I;

X^* — множество конечных последовательностей элементов X для любого множества X. Таким образом, $X^* = \{<x_1,...,x_n> \mid x_1,\ldots,x_n \in X\}$, для любого $n \geq 0$. Обратите внимание, что пустая последовательность находится в X^* для любого X.

4.2 Сокращения

В настоящем стандарте используются следующие сокращения:

CL — общая логика (Common Logic);

CG — концептуальный граф;

DF — форма отображения;

EBNF — формат расширенной формы Бэкуса-Наура (по аналогии с [2]);

FO — первый порядок (первого порядка);

KIF — формат обмена знаниями;

OWL — язык веб-онтологий;

RDF — структура описания ресурсов;

RDFS — схема структуры описания ресурсов;

TFOL — традиционная логика первого порядка;

XML — расширяемый язык разметки.

5 Требования и обзор структуры

5.1 Требования

5.1.1 Общая логика (Common Logic, CL) должна включать полную логику первого порядка с равенством.

Абстрактный синтаксис и семантика CL должны обеспечивать полный диапазон синтаксических форм первого порядка с обычными значениями. Любой традиционный синтаксис первого порядка будет напрямую переведен в CL без потери информации и изменения значения.

5.1.2 Общая логика должна поддерживать универсальный синтаксис для передачи логических выражений.

Общая логика должна поддерживать универсальный синтаксис для передачи логических выражений при соблюдении следующих условий:

а) для передачи содержимого CL через Интернет необходим единый синтаксис XML;

б) языки CL должны иметь возможность выражать различные часто используемые виды «синтаксического сахара» для логических форм и часто используемых шаблонов логических высказываний;

в) абстрактный синтаксис CL должен соответствовать существующим соглашениям; в частности, он должен быть способен отображать любое содержимое, выражаемое с помощью RDF, RDFS или OWL;

г) необходимо ввести по крайней мере один понятный человеку синтаксис представления, который можно использовать для выражения всего языка.

5.1.3 Общая логика должна быть простой и естественной для использования в сети Интернет.

Общая логика для использования в сети Интернет должна обладать следующими свойствами:

а) синтаксис XML должен быть совместим с опубликованными спецификациями XML, синтаксисом IRI, схемой XML, Unicode и другими соглашениями о передаче информации в Интернете;

б) в качестве имен в языке следует использовать IRI;

в) необходимо обеспечить возможность использования IRI в текстах заголовков и выражениях меток, чтобы облегчить выполнение действий в Интернете, например поиска, импорта и добавления перекрестных ссылок.

5.1.4 Общая логика должна поддерживать открытые сети.

Общая логика должна поддерживать открытые сети при соблюдении следующих условий:

а) передача содержимого между агентами, поддерживающими CL, не должна требовать согласования синтаксических ролей символов и перевода из одной синтаксической роли в другую;

б) все фрагменты текста CL должны иметь одинаковое значение и поддерживать одни и те же импликации в любой точке сети. Все имена должны иметь одинаковое логическое значение во всех узлах сети;

в) ни один агент не должен иметь возможность ограничивать способность другого агента ссылаться на какие-либо объекты или делать утверждения о чем-либо;

г) язык должен поддерживать способы обращения к локальной вселенной дискурса, а также иметь возможность соотносить ее с другими такими вселенными;

д) пользователи CL должны иметь право вводить новые имена и использовать их в опубликованном содержимом CL.

5.1.5 Общая логика не должна содержать произвольные допущения в отношении семантики.

Общая логика не должна содержать произвольные допущения в отношении семантики при следующих условиях:

а) CL не делает необоснованных или произвольных предположений о логических отношениях между различными выражениями;

б) если возможно, агенты CL должны выражать эти предположения напрямую в CL.

5.2 Семейство языков

В этом подразделе описано, что подразумевается под «семейством» языков, а также приведено определенное обоснование разработки CL.

В соответствии с соглашением, согласно которому любой язык должен иметь грамматику, CL представляет собой семейство языков, а не один язык. Различные языки CL, называемые в настоящем стандарте диалектами, могут существенно отличаться по внешнему синтаксису, но имеют единую унифицированную семантику и могут быть преобразованы в общий абстрактный синтаксис. Язык относится к семейству, если существует возможность взаимного перевода на другие диалекты и с них при условии сохранения значения. При этом не учитывается наличие какой-либо определенной синтаксической формы. Поэтому несколько существующих логических обозначений и языков можно рассматривать как диалекты CL.

В примерах в настоящем стандарте применен диалект CL CLIF на основе KIF (см. приложение А). CLIF можно рассматривать как обновленную и упрощенную формы KIF 3.0 [3] и, следовательно, как самостоятельный язык. Концептуальные графы [3] также являются хорошо известной формой логики первого порядка для машинной обработки; язык CGIF описан в приложении Б. На основании требований 5.1.2 а) и 5.1.3 а) в приложении В описан полностью совместимый диалект XML под названием XCL.

6 Абстрактный синтаксис и семантика общей логики

6.1 Абстрактный синтаксис общей логики

6.1.1 Категории абстрактного синтаксиса

6.1.1.1 Термины, маркеры последовательности, высказывания, утверждения и тексты — это правильно построенные выражения.

6.1.1.2 Текст — это текстовая конструкция, ограничение предметной области или импорт.

6.1.1.3 Текстовая конструкция содержит набор, список или комплект высказываний, утверждений и (или) текстов. Текст CL может иметь вид последовательности, набора или комплекта высказываний, утверждений и (или) текстов; диалекты могут указывать на подразумеваемые понятия или оставлять их без определения. Переупорядочивание и повторение аргументов текстовой конструкции не имеют значения с точки зрения семантики. Тем не менее приложения, которые передают или повторно публикуют текст CL, должны сохранять структуру текстовых конструкций, поскольку другие приложения могут использовать такую структуру в определенных целях, например для индексации. Если диалект налагает условия для текстовых конструкций, совместимые приложения должны сохранить эти условия. Текстовая конструкция может быть пустой.

6.1.1.4 Ограничение предметной области состоит из термина и текста, который называется основным текстом. Этот термин указывает на «локальную» вселенную дискурса, в рамках которой понимается текст.

6.1.1.5 Импорт содержит заголовок. Предполагается, что заголовок содержит идентификатор внешнего текста CL, а импорт повторно утверждает этот внешний текст в импортируемом тексте.

6.1.1.6 Аксиома — это утверждение, высказывание или текст.

6.1.1.7 Утверждение — это дискурсивное утверждение либо заголовок.

6.1.1.8 Дискурсивное утверждение — это утверждение вне дискурса либо утверждение в дискурсе.

6.1.1.9 Утверждение вне дискурса содержит последовательность терминов.

6.1.1.10 Утверждение в дискурсе также содержит последовательность терминов.

6.1.1.11 Заголовок содержит имя и текст. В ходе озаглавливания тексту присваивают заголовок. Заголовки часто имеют форму IRI, которые идентифицируют текст как ресурс.

6.1.1.12 Высказывание может быть количественным, логическим или простым.

6.1.1.13 Количественное высказывание имеет (i) тип, называемый квантором, (ii) конечную неповторяющуюся последовательность интерпретируемых имен, называемую связывающей последовательностью, каждый элемент которой называется обязательным условием количественного высказывания, и (iii) высказывание, называемое основной частью количественного высказывания. В абстрактном синтаксисе для количественного высказывания различают универсальный тип и тип существования. Имя, встречаемое в связывающей последовательности, называют связанным с основной частью высказывания. Все имена и маркеры последовательности, которые не связаны с основной частью высказывания, считают свободными.

6.1.1.14 Логическое высказывание имеет тип, называемый соединительным элементом, и количество высказываний, называемых компонентами логического высказывания. Количество зависит от конкретного типа. В абстрактном синтаксисе различают пять типов логических высказываний: конъюнкции и дизъюнкции, которые могут иметь любое количество компонентов, импликации и двойные импликации, которые имеют ровно два компонента, а также отрицания, которые имеют ровно один компонент. Два компонента импликации выполняют разные роли; один является антецедентом, а другой — консеквентом.

6.1.1.15 Простое высказывание — это уравнение, содержащее два аргумента, которые являются терминами, или атомарное высказывание, состоящее из термина, называемого предикатом, и последовательности терминов, называемой последовательностью аргументов, элементы которой называют аргументами атомарного высказывания.

6.1.1.16 Термин — это имя или функциональный термин. К термину может быть прикреплен комментарий. При этом каждое имя — это термин.

6.1.1.17 Функциональный термин состоит из термина, называемого оператором, и последовательности терминов, называемой последовательностью аргументов, элементы которой называют аргументами функционального термина.

6.1.1.18 Последовательность терминов — это конечная последовательность терминов и (или) маркеров последовательности. Последовательности терминов могут быть пустыми, но функциональный термин с пустой последовательностью аргументов не допускается отождествлять со своим оператором, а атомарное высказывание с пустой последовательностью аргументов нельзя отождествлять со своим предикатом.

6.1.1.19 Лексикон — это набор имен (т. е. словарь лексикона), набор маркеров последовательности и набор заголовков.

6.1.1.20 Нерегулярные высказывания в конкретном синтаксисе разделяются в абстрактный синтаксис как предложения (т. е. нульарные атомарные высказывания) с новым именем предиката. Таким образом, нерегулярные высказывания могут быть вложены в тексты, утверждения и (в остальном) обычные составные высказывания, а семантика таких выражений определяется, как правило, на основе таблицы 2.

6.1.1.21 Комментарий — это объект данных. К правильно построенным выражениям, которые являются текстами, утверждениями, высказываниями или функциональными терминами, можно добавить любое количество комментариев. Тем не менее, это не относится к именам и маркерам последовательности и другим комментариям. Особые ограничения относительно характера комментариев CL отсутствуют. В частности, комментарий может быть текстом CL. В некоторых диалектах могут действовать определенные ограничения на форму комментариев.

В 6.1 приведено полное описание абстрактной синтаксической структуры CL. Все полностью совместимые диалекты CL должны обеспечивать однозначное синтаксическое представление каждого из вышеуказанных типов правильно построенных выражений.

Типы высказываний обычно обозначают включением явных текстовых строк, таких как «forall» для универсального высказывания и «and» для конъюнкции. Однако ограничения относительно того, как различные синтаксические категории могут быть представлены в поверхностных формах диалекта, отсутствуют. В частности, выражения на диалекте не обязательно должны состоять из символьных строк.

6.1.2 Метамодель абстрактного синтаксиса общей логики

6.1.2.1 Имена и маркеры последовательности

Классы имен и маркеров последовательности в языке CL получены из строк с использованием следующих операторов:

- *Voc : String → V;*
- *Seqmark* : *String → Smark;*
- *Titling* : *String → Ttl;*
- *Binder = V ∪ Smark.*

6.1.2.2 Термины и последовательности терминов

Класс терминов в языке CL — это класс Term, который включает в себя все имена из V и все функциональные термины. Класс функциональных терминов в языке CL — это класс FunctionalTerm, создаваемый за счет рекурсивного применения оператора *Func*[1)] к парам, состоящим из одного термина и одной последовательности терминов. Класс последовательностей терминов в языке CL — это класс TermSequence, который включает все конечные последовательности терминов и/или маркеров последовательности.

- *Func Term* x *TermSequence → FunctionalTerm;*
- *Term = V ∪ FunctionalTerm;*
- *TSeq:\Term ∪ Smark)* x ... x *{Term ∪ Smark) → TermSequence.*

6.1.2.3 Высказывания

Класс высказывания в языке CL — это класс Sentence, который включает в себя все простые высказывания (включая уравнения, если таковые имеются), сформированные путем применения атомарных (и Id) операций из правильно построенных терминов и последовательностей терминов, а также всех составных высказываний, образованных путем рекурсивного применения набора операций Neg, Conj, Disj, Cond, BiCond, EQuant и UQuant, которые отвечают следующим условиям:

- каждая операция взаимно однозначна;
- наборы операций отделены попарно и не пересекаются с множеством терминов λ:
- *Atomic-Term* x *TermSequence → Sentence;*
- *Id: Term* x *Term → Sentence;*
- *Neg-.Sentence → Sentence;*
- *Conj : Sentence* x... x *Sentence → Sentence;*
- *Disj : Sentence* x ... x *Sentence → Sentence;*
- *Cond : Sentence* x *Sentence → Sentence;*
- *BiCond-.Sentence* x *Sentence → Sentence;*
- *EQuant-.Binder* x... x *Binder* x *Sentence → Sentence, n* ≥ 0;
- *UQuant.Binder* x ... x *Binder* x *Sentence → Sentence, n ≥ 0.*

---

1) *Здесь и далее по тексту названия классов, терминов, операций приведены с заглавной буквы курсивом.*

Ненормативная метамодель, описывающая отношения между синтаксическими категориями, представлена ниже на рисунках 1—9.

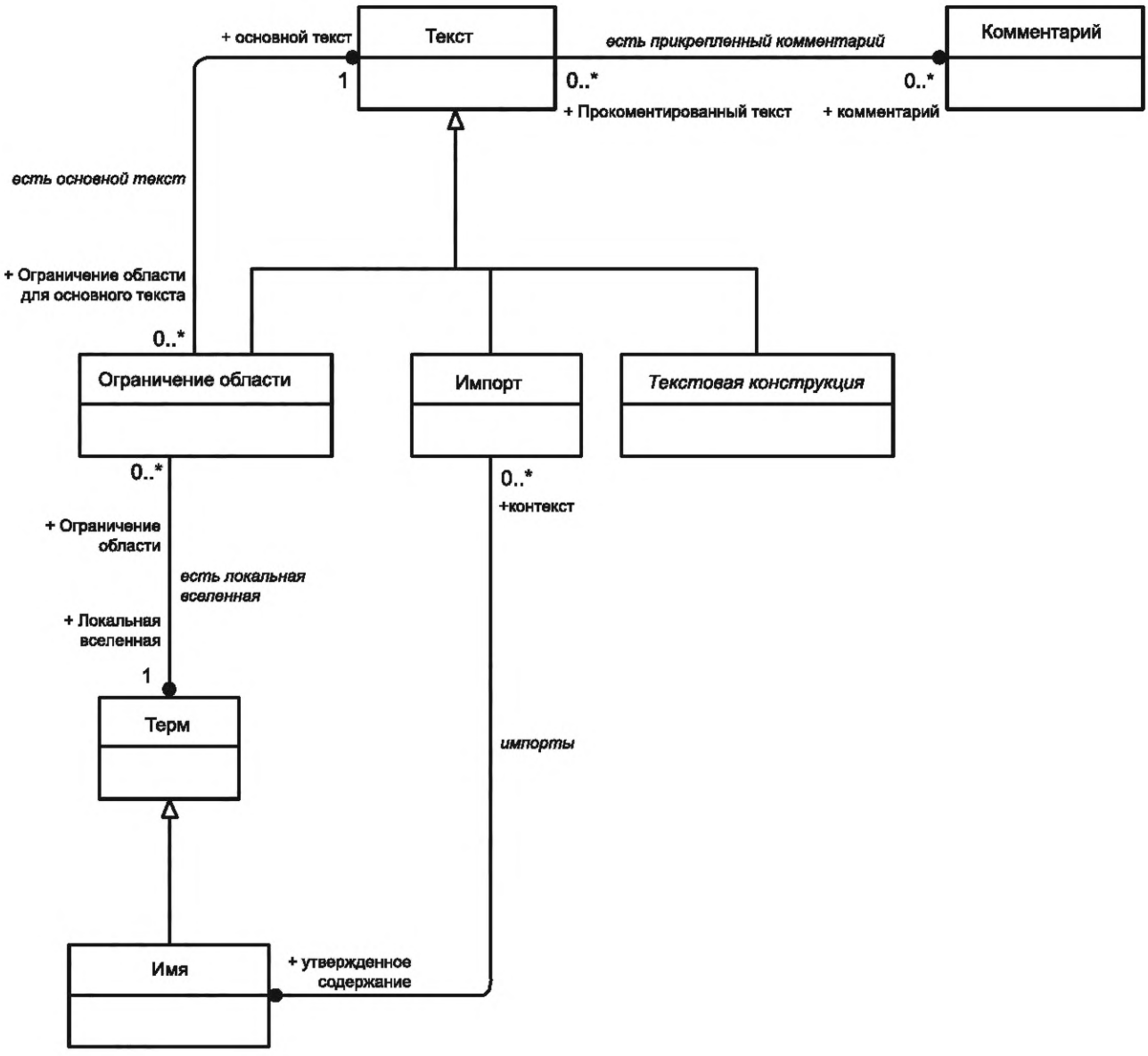

Рисунок 1 — Абстрактный синтаксис текстов[1)]

[1)] *«Термин» (3.16) здесь и далее в рисунках приведен в сокращении «терм».*

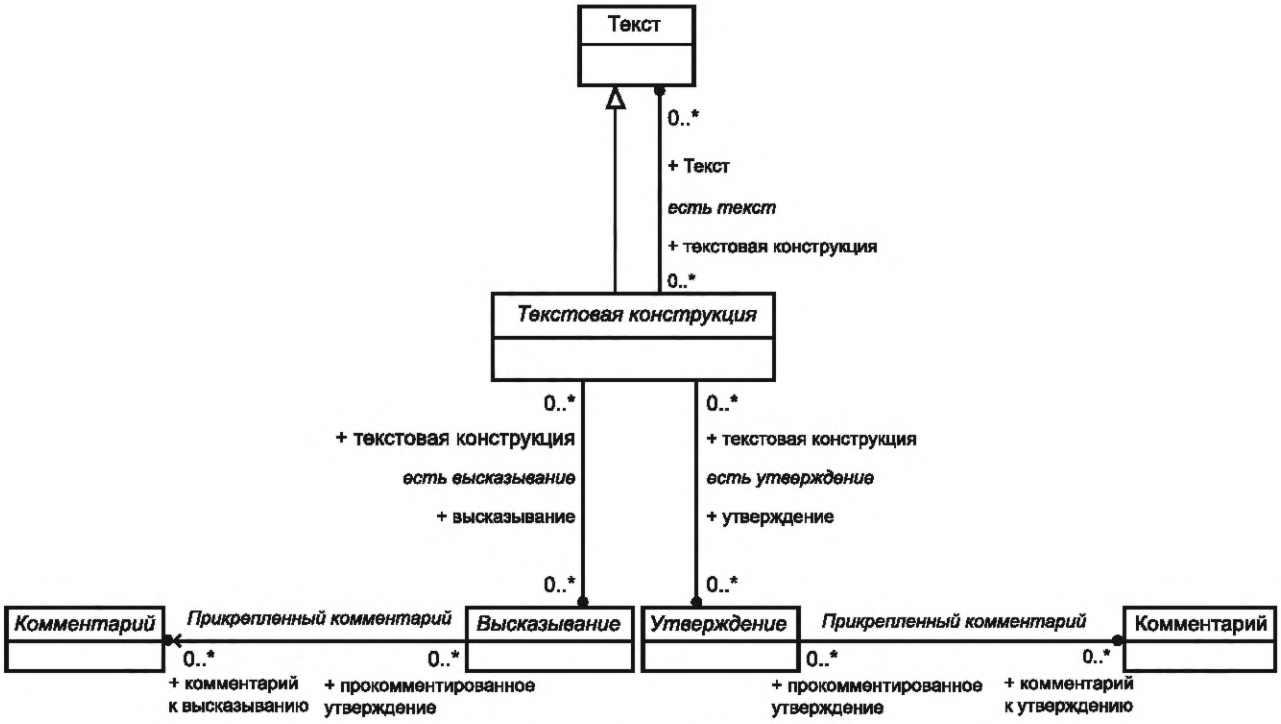

Рисунок 2 — Абстрактный синтаксис текстов

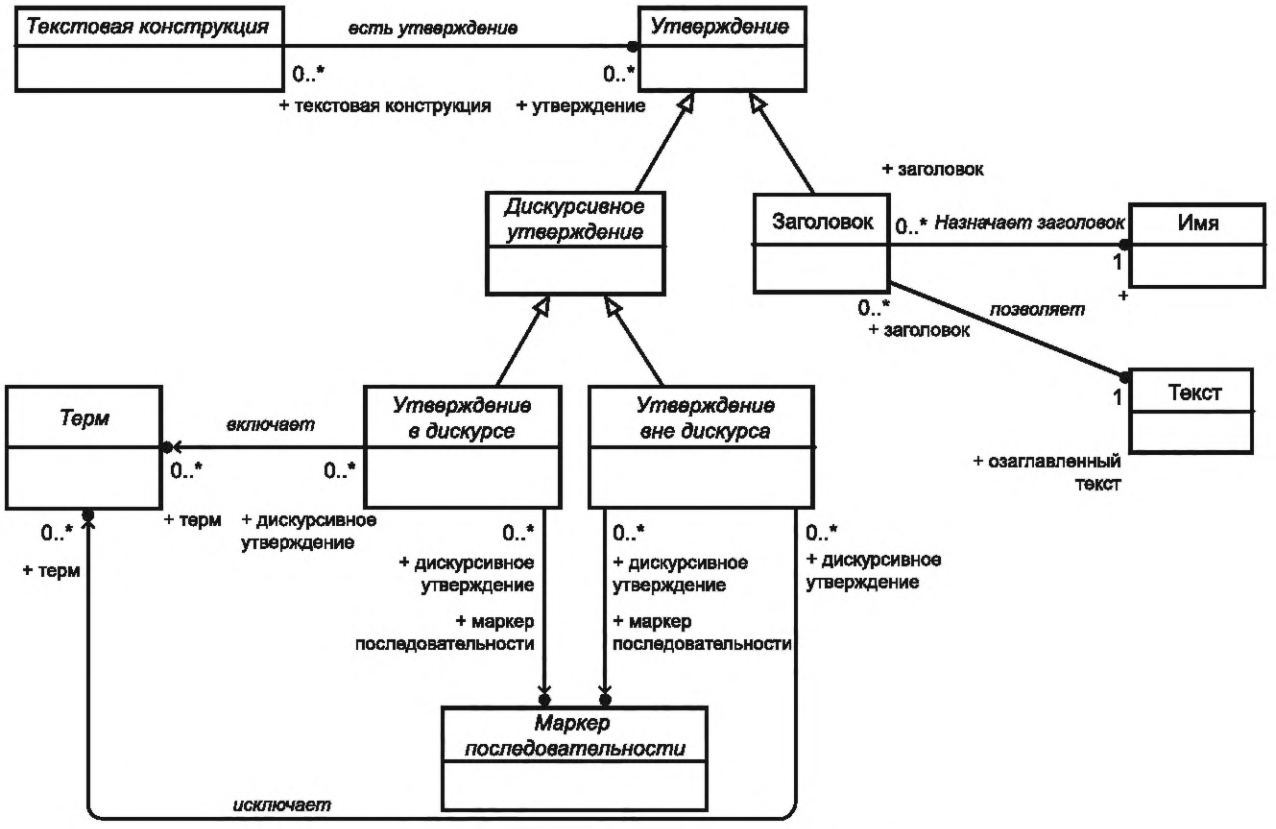

Рисунок 3 — Абстрактный синтаксис утверждений

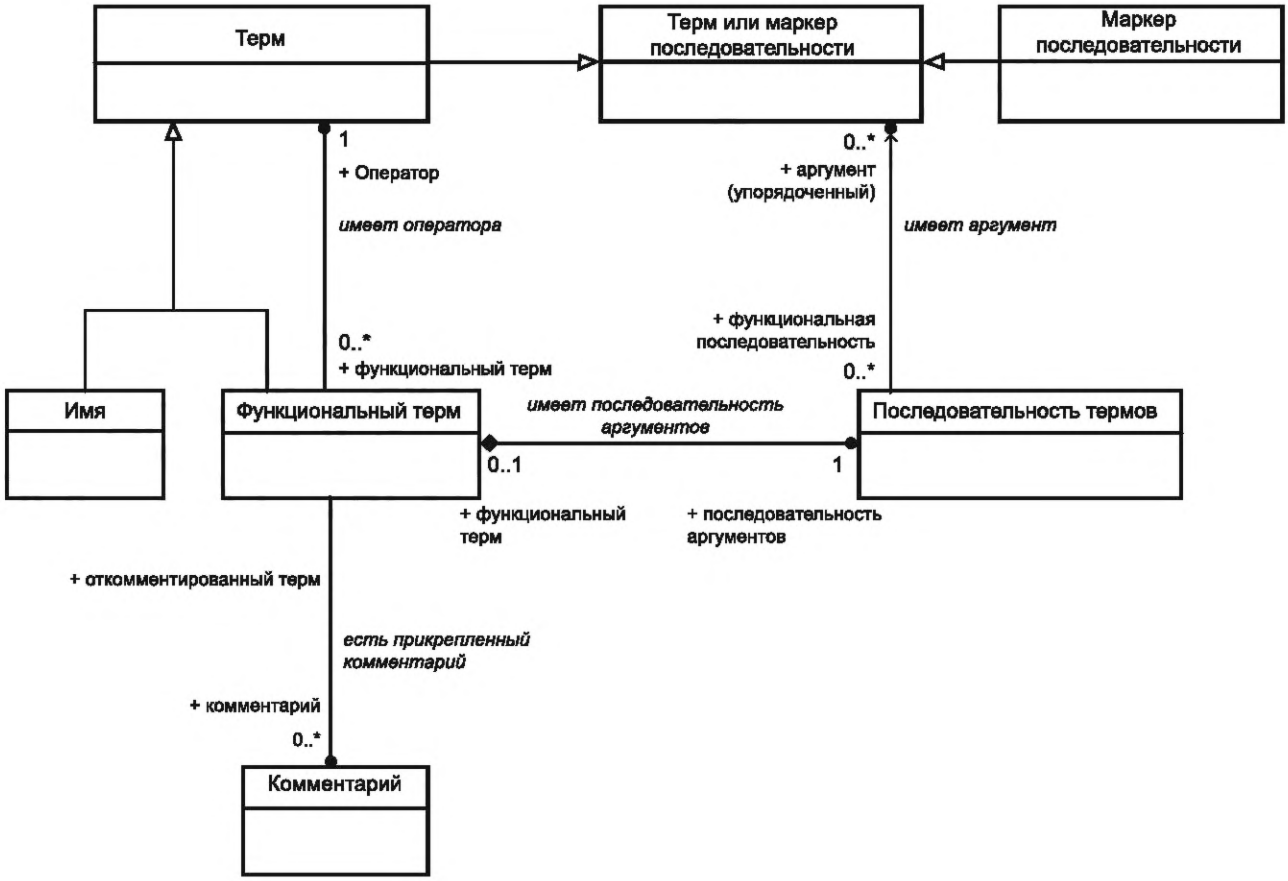

Рисунок 4 — Абстрактный синтаксис терминов (термов)

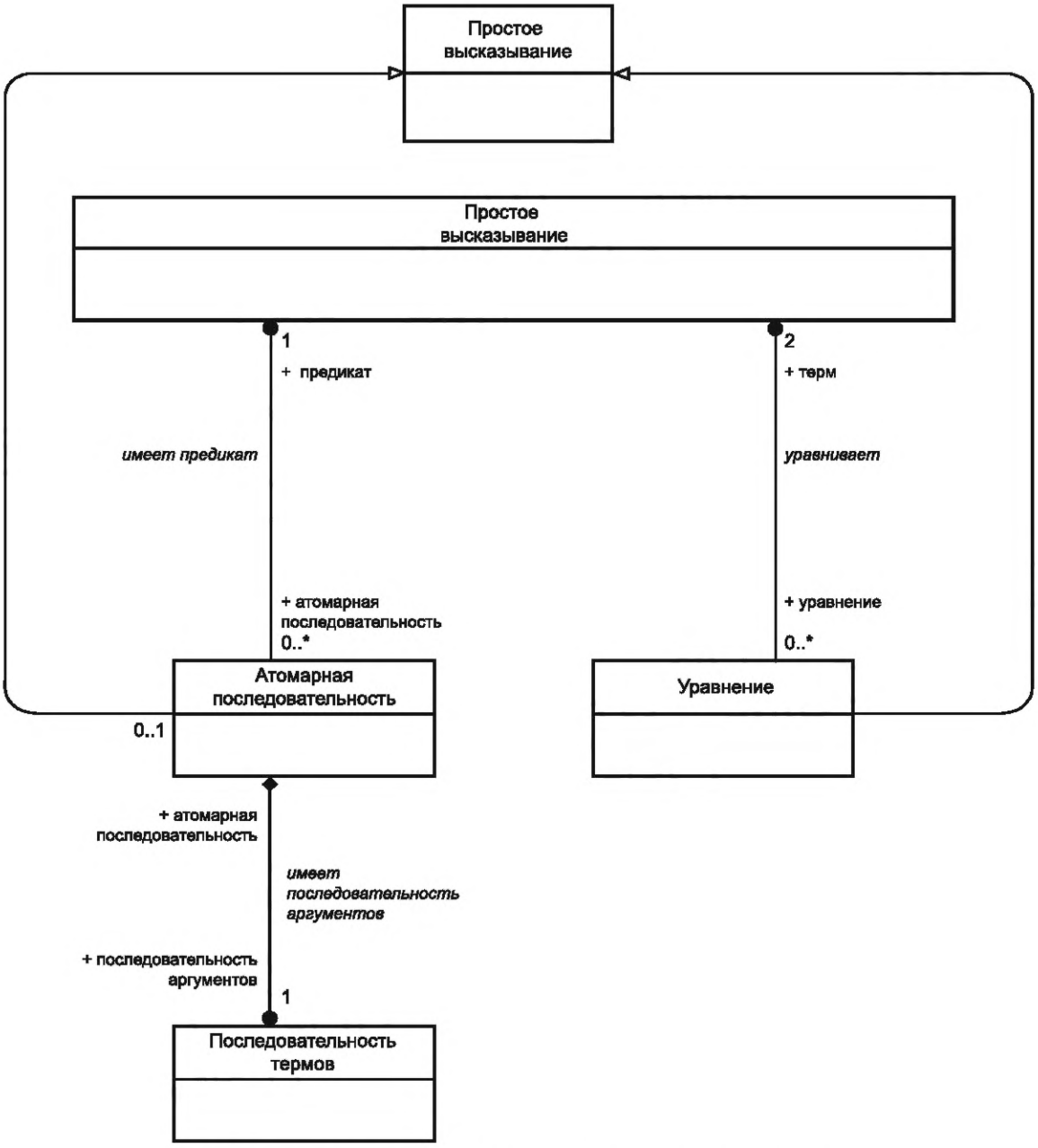

Рисунок 5 — Абстрактный синтаксис простого высказывания

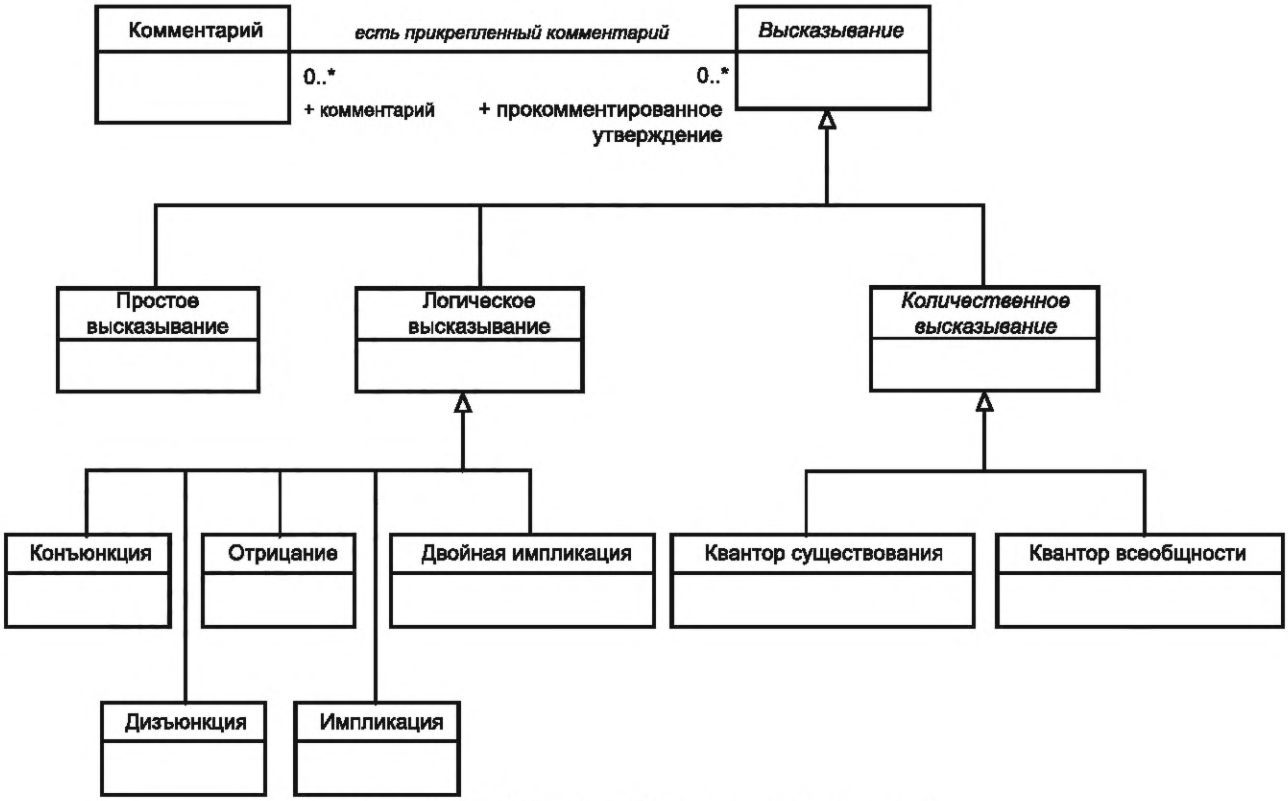

Рисунок 6 — Абстрактный синтаксис высказываний

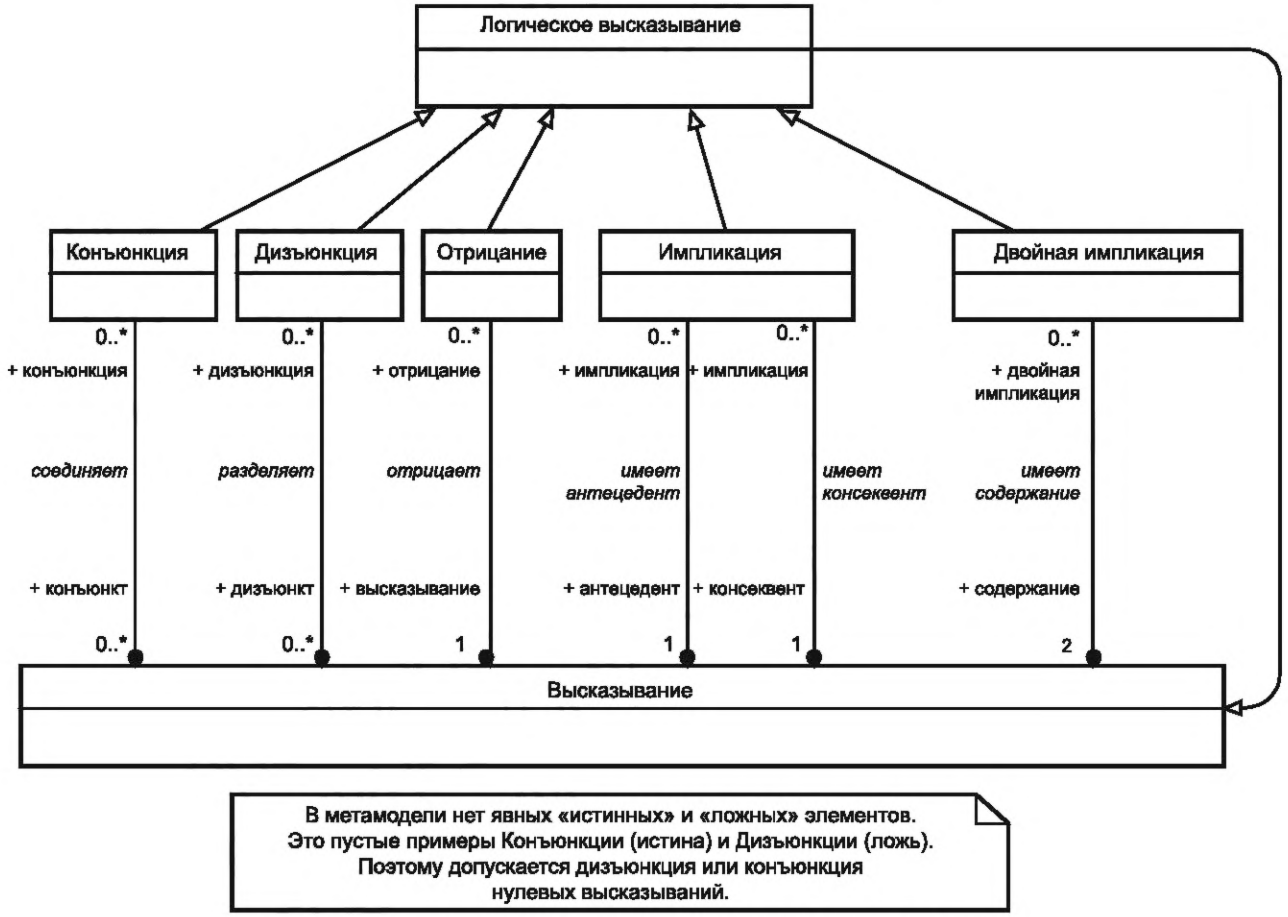

Рисунок 7 — Абстрактный синтаксис логических высказываний

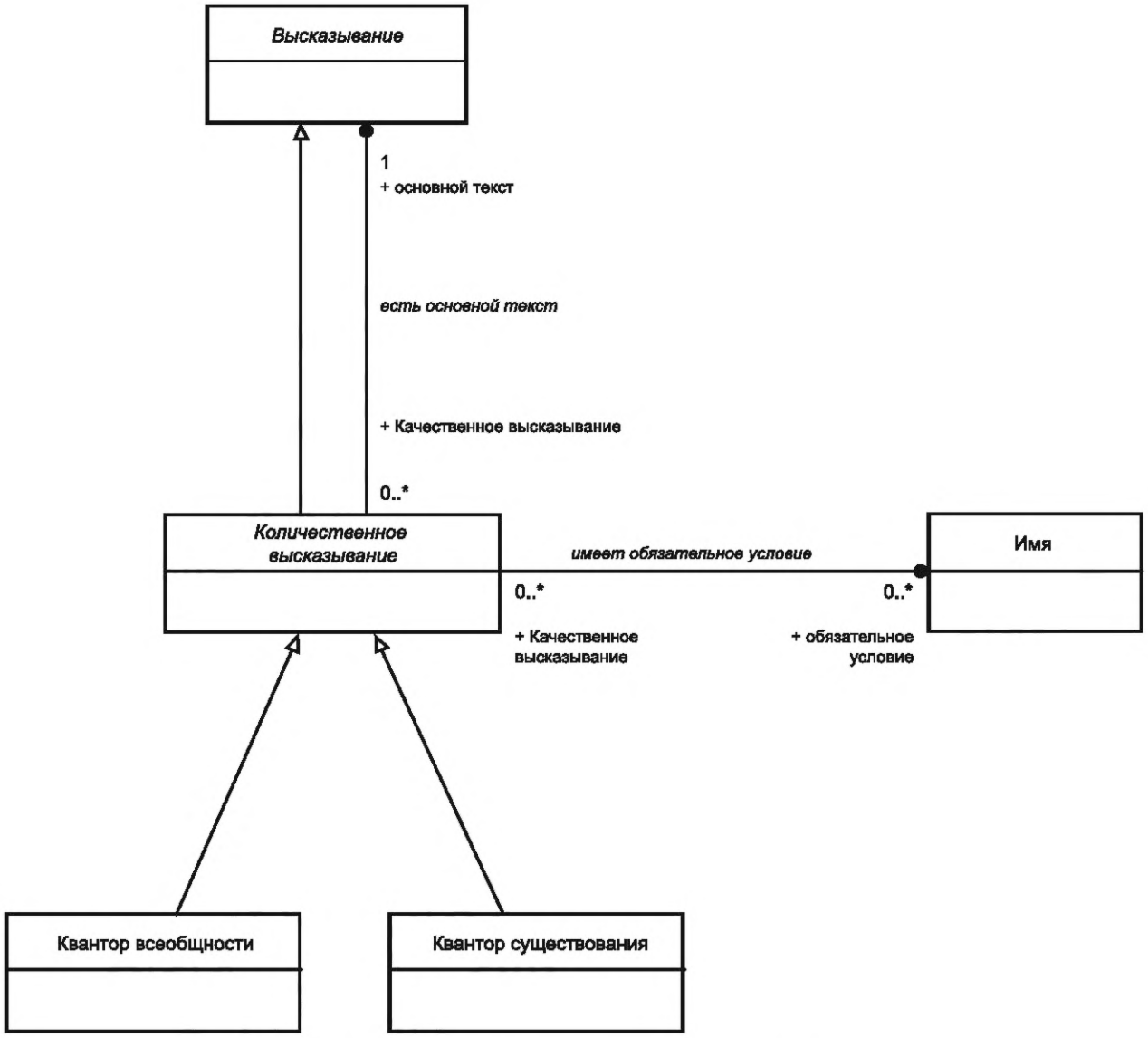

Рисунок 8 — Абстрактный синтаксис количественного высказывания

6.1.2.4 Утверждения

Класс утверждений в языке CL — это класс Statement, полученный в результате рекурсивного применения операций *outDiscourse, inDiscourse* и *title* в следующих условиях:

- *outDiscourse-.TermSequence → DiscourseStatement;*
- *inDiscourse-.TermSequence → DiscourseStatement;*
- *title* :Ttl x *Text → Titling.*

Дискурсивные утверждения и заголовки — это высказывания:

- *Statement = DiscourseStatement ∪ Titling.*

6.1.2.5 Тексты

Класс текстов в языке CL — это класс Text, полученный в результате рекурсивного применения набора операций *txt, imports* и *domain* в следующих условиях:

- *txt:[Sentence ∪ Statement ∪ Text)* x ... x*(Sentence ∪ Statement ∪ Text) → TextConstruction;*
- *imports*: Ttl → *Importation;*
- *domain:Term* **x** *Text → DomainRestriction.*

Текстовые конструкции, ограничения области и импорты — это тексты:

- *Text = TextConstruction ∪ DomainRestriction ∪ Importation.*

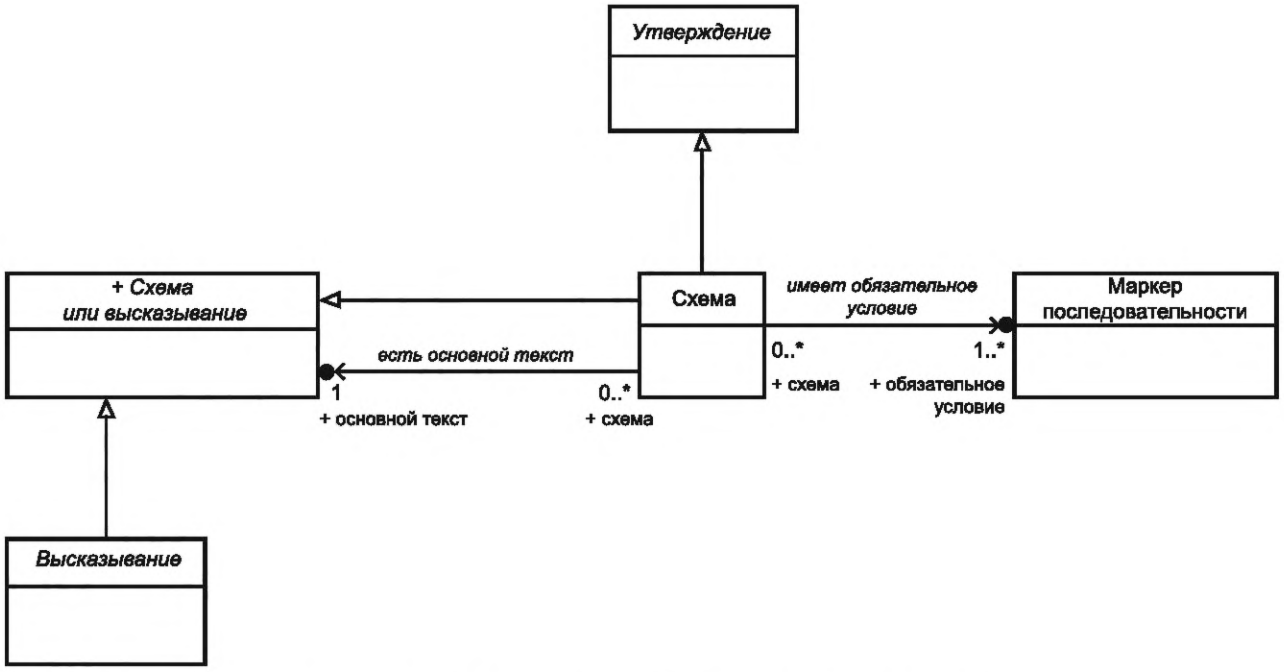

Рисунок 9 — Абстрактный синтаксис текстовых конструкций

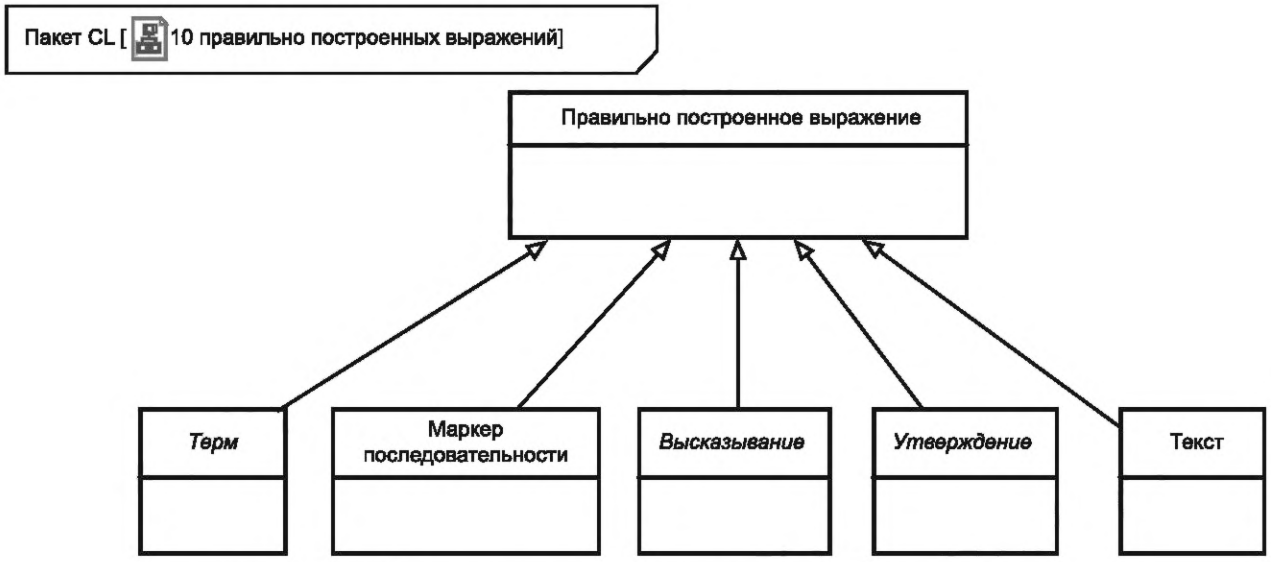

Рисунок 10 — Абстрактный синтаксис правильно построенных выражений

6.1.2.6 Правильно построенные выражения

Термины, маркеры последовательности, высказывания, утверждения и тексты — это правильно построенные выражения:

- *Wfe = Term ∪ SequenceMarker ∪ Sentence ∪ Statement ∪ Text.*

6.1.2.7 Метамодель

На рисунках с 1 по 9 представлена ненормативная метамодель, описывающая отношения между синтаксическими категориями.

6.1.3 Закрытие импорта

Импорт без заголовка в тексте — это выражение импорта, которое не было получено из заголовка в абстрактном синтаксическом дереве согласно описанию в п. 6.1.2.

Корпус — это набор текстов с абстрактным синтаксисом CL. Корпус может быть пустым, конечным или бесконечным. Фиксированная точка импорта при сопоставлении заголовка ttl — это корпус C.

При этом если G является текстом в C, то C содержит каждый текст G', полученный из G путем замены одного экземпляра импорта без заголовка с именем N на текст, который является значением ttl(N) сопоставления заголовка. Закрытие импорта корпуса C при сопоставлении заголовка ttl — это пересечение всех фиксированных точек импорта в ttl, которые содержат C. Закрытие импорта каждого непустого корпуса в языке CL не является пустым.

6.1.4 Структура абстрактного синтаксиса диалектов

Диалекты могут, помимо прочего, допускать другие формы построения предложений, не описываемые этим синтаксисом, но для обеспечений полной совместимости такие конструкции должны быть либо новыми категориями, определенными в терминах этих категорий, либо расширениями таких категорий (например, новыми видами логического высказывания или квантора), которые равны по значению конструкции, использующей только этот синтаксис и интерпретируемой в соответствии с семантикой CL. Иными словами, их можно рассматривать как систематические сокращения или макросы, также известные как «синтаксический сахар». Диалекты CLIF (описаны в приложении A) и XCL (описаны в приложении B) содержат ряд форм «синтаксического сахара» для количественных и атомарных высказываний. (Также допускают другие типы совместимости: полный отчет по совместимости см. в разделе 7).

Единственные неопределенные термины в выражении абстрактного синтаксиса — это имя, маркер последовательности и заголовок. Единственное необходимое синтаксическое ограничение для основных лексических категорий имени и маркера последовательности заключено в том, что они должны быть исключительными. Диалекты, предназначенные для передачи содержимого по сети, не должны налагать произвольные или необязательные ограничения на форму имен. Вместо этого они должны предусматривать использование определенных имен в качестве заголовков текстов CL. Диалекты, предназначенные для использования в Интернете, должны позволять использовать интернационализированные идентификаторы ресурсов в качестве имен [4], [5]. Диалекты CL должны определять имена в терминах соглашений Unicode [6].

В абстрактном синтаксисе CL отсутствует понятие «связанная переменная». Связанные имена не обязаны лексически отличаться от имен, которые могут быть (только) свободными. Кроме того, отсутствует требование разделять имена на различные классы, такие как отношения, функции и отдельные имена. Ограничения по сортировке имен отсутствуют. Определенные диалекты CL могут включать те или иные различия между подклассами имен и вводить дополнительные ограничения на появление типов имен или терминов в выражениях. Например, в них может быть требование, чтобы имена, которые могут быть связанными (т. е. переменные традиционных языков первого порядка), имели специальные префикс, как в KIF, или определенный стиль, как в Prolog. Также может быть необходимо, чтобы операторы находились в особой категории имен отношений (как в традиционном синтаксисе первого порядка).

Диалект может содержать определенные семантические условия для некоторых категорий имен и синтаксические ограничения относительно мест, в которых такие имена могут встречаться в выражениях. Например, синтаксисы CLIF и XCL рассматривают численные величины как имеющие фиксированное обозначение и запрещают их использование в качестве заголовков.

Диалект может требовать, чтобы некоторые имена были синтаксическими недискурсивными и никогда не должны обозначать сущности во вселенной дискурса. Это требование может быть внедрено, например, путем разбиения словаря или введения условия о том, что имена, встречающиеся в определенных синтаксических позициях, должны быть недискурсивными. Диалект с синтаксическими недискурсивными именами называют сегрегированным. В таких диалектах имена, которые не являются недискурсивными, называют дискурсивными именами.

Диалект должен содержать достаточные синтаксические ограничения, гарантирующие, что в любом синтаксически допустимом тексте диалекта:

- каждое имя классифицируют как дискурсивное либо как недискурсивное;
- имена не могут быть одновременно классифицированы как дискурсивные и недискурсивные;
- недискурсивные имена не могут быть аргументами простого предложения или функционального термина;
- недискурсивные имена не могут быть связанными в количественном высказывании.

Диалект может иметь дополнительные механизмы для встраивания информации в тексты CL, которые можно удалить, не затрагивая анализ текста с точки зрения абстрактного синтаксиса, а также опустить при переводе на определенные другие диалекты.

6.2 Семантика общей логики

Семантика CL определена в терминах как отношение удовлетворенности между текстом CL и математическими структурами, называемыми интерпретациями.

Интерпретация I в языке CL L с лексиконом λ = (Vλ, Smarkλ, Ttlλ) (где Vλ ⊆ V Smarkλ ⊆ Smark, Ttlλ ⊆ Ttl) является набором UR/, исходная вселенная с выделенным подмножеством UD/, вселенной дискурса и пятью сопоставлениями:

- int_I между именами в $V\lambda$ и UR/;
- rel_I между UR/ и множеством всех подмножеств UD/*;
- fun_I между UR/ и всюду определенными функциями от UR/* до UR_I;
- seq_I между маркерами последовательности в $Smark\lambda$ и UR_I*;
- *сопоставление заголовка* ttl_I от Ttl_λ до текстов L.

Если UD_I не является пустым, а диапазон fun_I представляет собой всюду определенные функции от UD_I* до UD_I, можно сказать, что / является основной интерпретацией L.

Интуитивно, UD_I — это вселенная или область дискурса, содержащая все индивидуальные элементы, о которых «рассказывает» интерпретация и которые охватывают кванторы. UR_I — это потенциально более широкий набор элементов, который также может содержать объекты, не входящие во вселенную дискурса. В частности, UR_I может содержать отношения, не входящие в UD_I, чтобы выступать в качестве интерпретации недискурсных имен. Все имена интерпретируют одинаково, независимо от того, воспринимают ли их как обозначающие что-то во вселенной дискурса. Поэтому существует только одно сопоставление интерпретации, которое применяют ко всем именам независимо от их синтаксической роли. В частности, rel_I (x) входит в UD_I, даже если x не входит в UD_I. При рассмотрении только классических диалектов элементы исходной вселенной, которые находятся за пределами вселенной дискурса, могут быть идентифицированы с их соответствующими значениями сопоставлений rel_I и fun_I, которые затем интерпретируют как сопоставление идентичности. Полученная конструкция напрямую сопоставляет предикаты с отношениями и операторы с функциями, создавая более традиционную структуру интерпретации для сегрегированного синтаксиса традиционной логики первого порядка.

Хотя маркеры последовательностей сопоставляют с конечными последовательностями в интерпретации, эти последовательности не обозначают именами, поэтому не должны обязательно входить в исходную вселенную.

Присвоение семантических значений сложным выражениям (в частности, присвоение значений истинности высказываниям) требует некоторых вспомогательных определений. Спецификация вспомогательных определений, используемых в семантике, приведена в таблице 1.

Пусть S — это подмножество $V \cup Smark$. Интерпретация J V — это S-версия I, которая точно повторяет I, за исключением того факта, что int_I и seq_I могут отличаться от int_J и seq_J с точки зрения того, что связывают с членом S. Более формально, / — это S-версия I if $UR_j = UR_I$, $UD_j = UD_I$, $rel_j = rel_I$, $fun_j = fun_I$, $ttl_I = ttl_j$, $int_j(n) = int_I(n)$ для имен $n \notin S$, $int_I(n) \in UD_j$ для имен $n \in S$, $seq_j(s) = seq_I(s)$ для маркеров последовательности $s \notin S$ и $seq_j(s) \in UD^*_j$ for $s \in S$.

Если E является подмножеством UD_I, то *ограничение* I до E является интерпретацией K того же языка L в той же исходной вселенной, где $int_K = int_I$ и $seq_K = seq_I$, но $UD_K = E$, $rel_K(v)$ является ограничением rel_I (v) до E* и $fun_K(v) = fun_j(v)$ для всех v в словаре I.

Если s = <s1, ..., s_n> и t = <t1, ..., t_m> — это конечные последовательности, тогда s;t — это составная последовательность <s_1, ..., s_n, t_1, ..., t_m>. В частности, s;<> = s для любой последовательности s.

Таблица 1 — Спецификация вспомогательных определений, используемых в семантике

$E \in \lambda$	ArgC(E) = ∅
$E = Func(T,T1,...,T_n)$	ArgC(E) = { T_1,…,T_n } ∪ ArgC(T)
E = Atomic(T,T1,…,T_n)	ArgC(E) = { T_1,…,T_n } ∪ ArgC(T)
E = Neg(S)	ArgC(E) = ArgC(S)
Примечание — В этой таблице указан набор констант аргументов ArgC (E) для E во всех правильно построенных выражениях E	
E = Con(S_1…S_n)	ArgC(E) = ArgC(S_1…S_n)

Окончание таблицы 1

E = EQuant(N,S)	*ArgC(E) = ArgC(S)* ∪ *N*
E = outDiscourse(T_1,...,T_n)	ArgC(E) = U_i ArgC(T_i)
E = inDiscourse(T_1,...,T_n)	ArgC(E) = U_i ArgC(T_i)
E = txt(E_1,...,E_n)	*ArgC(E) = ArgC(E_i)*
E = domain(T,G)	*ArgC(E) = ArgC(G)* ∪ *ArgC(T)*
E = title(E_1,G)	ArgC(E) = ArgC(G)
E = imports(E_1)	ArgC(E) = ∅
Примечание — В этой таблице указан набор констант аргументов ArgC (E) для E во всех правильно построенных выражениях E.	

Значение любого выражения E в интерпретации I задают в соответствии с правилами, приведенными в таблице 2.

Таблица 2 — Интерпретации выражений CL

E1	Имя N	I(E) = int_I(N)
E2	Маркер последовательности S	I(E) = seq_I (S)
E3	Заголовок N	I(E) = ttl_I (N)
E4	Последовательность терминов T_1 … T_n с T_1 в качестве терма	I(E) = <$_I$ (T_1)>; $_I$ (<T_2 ... T_n >)
E5	Последовательность терминов T_1 … T_n с T_1 в качестве маркера последовательности	I(E) = $_I$ (T_1); $_I$ (<T_2 ... T_n>)
E6	Терм с оператором O и последовательностью аргументов S	I(E) =fun_I ($_I$ (O))($_I$ (S))
E7	Простое высказывание, которое представляет собой уравнение с терминами T_1, T_2	I(E) = true, если I(T_1) = I(T_2); в противном случае I(E) = false
E8	Атомарное высказывание с предикатом P и последовательностью аргументов S	I(E) = true, если I(S) находится в rel_I(I(P)); в противном случае I(E) = false
E9	Логическое высказывание типа «отрицание» с компонентом C	I(E) = true, если I(C) = false; в противном случае I(E) = false
E10	Логическое высказывание типа «конъюнкция» с компонентами C_1 … C_n	I(E) = true, если I(C_1) = … = I(C_n) = true; в противном случае I(E) = false
Ell	Логическое высказывание типа «дизъюнкция» с компонентами C_1 … C_n	I(E) = false, если I(C_1) = … = I(C_n) = false; в противном случае I(E) = true
E12	Логическое высказывание типа «импликация» с компонентами C_1, C_2	I(E) = false, если I(C_1) = true и I(C_2) = false; в противном случае I(E) = true
E13	Логическое высказывание типа «двойная импликация» с компонентами C_1, C_2	I(E) = true, если I(C_1) = I(C_2); в противном случае I(E) = false
E14	Количественное высказывание типа «вселенная» с обязательными условиями N и основным текстом B	I(E) = true, если для каждой N-версии J в I, J(B) является истинным; в противном случае I(E) = false
E15	Количественное высказывание типа «существование» с обязательными условиями N и основным текстом B	I(E) = true, если для части N-версий J в I, J(B) является истинным; в противном случае I(E) = false

Окончание таблицы 2

E16	Нерегулярное высказывание S	I(E) = int_I (S)
E17	Утверждение вне дискурса outDiscourse(T_1 …T_n)	I(E) = true, если I (T_i) ∉ UD_I U UD_I* для 0 < i < n; в противном случае I(E) = false
E18	Утверждение в дискурсе inDiscourse(T_1 … T_n)	I(E) = true, если I (T_i) ∈ UD_I U UD_I * для 0 < i < n; в противном случае I(E) = false
E19	Текстовая конструкция txt(E_1 … E_n)	I(E) = true, если I(E_1) = … = I(E_n) = true; в противном случае I(E) = false
E20	Ограничение области domain(N,G)	I(E) = true, если присутствует интерпретация J = [I<{x||x|∈ rell/(I(N))}] и J(G) = true; в противном случае I(E) = false
E21	Заголовок текста (E1, G), где G — текст в смысле абстрактного синтаксиса.	I(*E*) = true, если ttll(E1) = G; в противном случае I(*E*) = false
E22	Утверждение импорта imports(E1)	I(*E*) = true

Это основные логико-семантические условия, которым должны удовлетворять все совместимые диалекты. Для текстов, в которых встречаются интерпретируемые имена, действуют дополнительные ограничения на интерпретацию (см. 6.3). Диалект может налагать дополнительные семантические ограничения помимо указанных.

Семантическое расширение, которое ограничивает интерпретации CL в соответствии с соглашениями о наименованиях, например соглашениями об идентификации сети, называют внешним. Внешние семантические ограничения могут иметь отношение к соглашениям или структурам, определенным вне теории модели.

В таблице 2 не описана интерпретация комментариев. Интерпретация выражения с прикрепленным комментарием аналогична интерпретации соответствующего выражения без комментария. Таким образом, добавление и удаление комментариев не затрагивает истинность любого текста CL. Тем не менее комментарии являются частью формального синтаксиса, поэтому приложения должны сохранять их при передаче, редактировании и повторной публикации текстов CL. В частности, считают, что имя, используемое в заголовке текста CL, сопоставляется с выражением абстрактного синтаксиса, поэтому при использовании того же имени в том же корпусе в качестве заголовка текста, который анализируют с помощью другого абстрактного выражения синтаксиса, данный корпус будет невыполнимым (см. 6.4), даже если тексты, за исключением комментариев, идентичны.

Заголовки не должны абсолютно точно соответствовать документам, файлам и другим единицам хранения данных. Диалекты и имплементации могут предусматривать распределение текстов по блокам хранения или хранение нескольких текстов с именами в одном блоке. Соглашения о заголовках в тексте могут быть связаны с соглашениями об адресации, используемыми для блоков данных, но это условие не является обязательным. Тексты также можно идентифицировать на основании внешних соглашений об именах, например путем кодирования текста в документах или файлах, имеющих сетевые идентификаторы. Семантика CL, описанная в 6.2, должна применяться ко всем идентификаторам сети, используемым в качестве заголовков в той сети, где публикуют или передают тексты CL.

6.3 Типы данных

Тип данных — это сопоставление лексического пространства (которое может быть явно представлено в синтаксисе) с пространством значений (которое является произвольным). В абстрактном синтаксисе элементы лексического пространства типа данных являются интерпретированными именами.

Семантика типов данных в CL зависит от описанных ниже условий:

- денотат интерпретированных имен одинаков во всех интерпретациях;
- денотаты интерпретированных имен типов данных по умолчанию определены явным образом;
- денотаты интерпретированных имен стандартизованных типов данных должны быть определены соответствующим стандартом (например, XSD);
- для типов данных, определяемых пользователем, денотаты должны быть указаны посредством явной аксиоматизации или определения на математическом языке либо с помощью механизмов для

определяемых пользователем типов данных, предусмотренных в других стандартах (например, XML Schema или Relax NG). Если именем типа данных является IRI, то этот IRI должен разыменовать документ, который предоставляет это определение.

Появление любого интерпретированного имени в тексте CL накладывает ограничение на интерпретацию этого текста, поскольку значение int(I) для такого имени всегда является истинным значением, присвоенным типом данных, связанным с этим именем. Диалект CL, который включает интерпретированные имена в соответствии с приведенными выше спецификациями, не рассматривают как семантическое расширение.

6.4 Удовлетворенность, действительность и логическое следствие

Поскольку семантика CL не предполагает строгого различия между именами вне дискурса и в дискурсе, тексты в этих двух наборах могут иметь различия. Суппозиция дискурса может быть полной или частичной. Полная пресуппозиция дискурса D — это разделение лексикона λ имен (V) и маркеров последовательности (Smark) на наборы имен и маркеров последовательности λ_D, которые находятся в дискурсе, а также на дополнительный набор имен и маркеров последовательности λ_N вне дискурса. В то время как полная пресуппозиция дискурса позволяет назначить дискурс каждому символу в лексиконе, частичная пресуппозиция обеспечивает назначение дискурса для соответствующего подмножества лексикона; $\lambda \cup D \cup \lambda_N$ — подмножество λ.

Интерпретация I соответствует пресуппозиции дискурса D, только если:

а) для любого имени $N \in \lambda_D$ существует $int_I(N) \in UD_I$;

б) для любого маркера последовательности $S \in \lambda_D$ существует $seq_I(S) \in UD_I^*$;

в) для любого имени $N \in \lambda_N$ существует $int_I(N) \notin UD_I$;

г) для любого маркера последовательности $S \in \lambda_N$ существует $seq_I(S) \notin UD_I^*$.

Корпус C CL выполняется интерпретацией I при наличии пресуппозиции дискурса D, только если:

- $I(G) = true$ и $ArgC(G) \in UD_I \cup UD_I^*$ для каждого G при закрытии импорта C в ttl_I;
- I соответствует D.

Корпус C выполняется при наличии пресуппозиции дискурса D, если существует базовая интерпретация, которая удовлетворяет ему и соответствует D. В противном случае корпус невыполним или противоречив. Если каждая базовая интерпретация, удовлетворяющая S при наличии пресуппозиции дискурса D, также удовлетворяет C при наличии той же пресуппозиции дискурса, то в этом случае S логически подразумевает C.

Корпус C выполним, если существует интерпретация, которая ему удовлетворяет; в противном случае корпус невыполним или противоречив. Если каждая интерпретация, удовлетворяющая S, также удовлетворяет C, то S логически подразумевает C.

Интерпретации CL воспринимают нерегулярные высказывания как непрозрачные переменные высказывания и требуют, чтобы такие высказывания сопоставлялись с предложениями (нульарными атомарными высказываниями) в абстрактном синтаксисе. В диалекте, который распознает нерегулярные высказывания, приведенные выше определения используют для обозначения интерпретаций, формируемых семантикой диалекта. Тем не менее, когда их классифицируют с помощью префикса прилагательного или наречия «common-logic», например «common-logic entails», следует рассматривать их как часть интерпретаций, которые точно соответствуют семантическим условиям CL. Например, диалект может поддерживать модальные предложения, а его семантика поддерживает следствие (необходимое P) подразумевает P, но это следствие не характерно CL, даже если бы язык был совместим с расширением CL. Однако следствие (Необходимое P) подразумевает (Необходимое P) характерно для CL.

В некоторых более поздних обсуждениях рассматривают ограниченные классы интерпретаций. Все приведенные выше определения могут применять только к интерпретациям определенного ограниченного класса. Таким образом, S *foo* подразумевает T, когда в любой интерпретации I в классе *foo* I удовлетворяет S, а I удовлетворяет T. Следование (или неудовлетворенность) по отношению к классу интерпретации подразумевает следование (или неудовлетворенность) по отношению к любому подмножеству этого класса.

При описании вывода T из S такое S упоминают как множество предпосылок, а T — как вывод следствия.

6.5 Маркеры последовательности, рекурсия и списки аргументов: обсуждение

Маркеры последовательности выводят CL за пределы экспрессивности первого порядка. Маркер последовательности, встречающийся в последовательности аргументов, означает произвольную конечную последовательность аргументов. Универсальное высказывание, связывающее маркер последовательности, имеет тот же семантический смысл, что и бесконечная конъюнкция всех выражений, полученных путем замены маркера последовательности конечной последовательностью имен, связанных квантором всеобщности.

Эта способность представлять бесконечные наборы высказываний в конечной форме означает, что CL с маркерами последовательностей не является компактным языком и, следовательно, не относится к первому порядку. Очевидно, что бесконечный набор высказываний, соответствующих по значению одному высказыванию, количественно определяющему маркер последовательности, логически эквивалентен этому высказыванию и, следовательно, логически подразумевает его (в отличие от конечных подмножеств бесконечного множества). Однако высказывания, содержащие маркеры последовательности, следует использовать как схемы аксиом. Когда они ограничиваются этой функцией, это приводит к получению компактной логики. С учетом этого маркеры последовательности на верхнем уровне утверждения текста могут быть связаны только кванторами всеобщности, а высказывания можно использовать только в качестве аксиом. Это ограничение часто подходит для текстов, которые считают «онтологиями», т. е. авторитетными источниками информации, представляющими концептуализацию определенной области приложения, предназначенной для применения к другим данным.

Компактный диалект, который не поддерживает маркеры последовательности, может имитировать большую часть функций, обеспечиваемых маркерами последовательности за счет использования списков однозначных аргументов, представленных в CL терминами, созданными на основе функции построения списка. Маркер последовательности преобразуется в имя списка, а квалификатор по именам списков заменяет квалификатор по маркерам последовательности. Тогда условие конечности последовательностей соответствует неоднозначному предположению о фиксированной точке для всех «стандартных» моделей аксиом списков. Такие соглашения широко используют в логическом программировании, а также в RDF и OWL. Этот метод вызывает значительное снижение синтаксической ясности и удобочитаемости, вынуждает допускать списки в качестве объектов в области дискурса и, в некоторых случаях, требует использования внешнего программного обеспечения для управления списками. Преимущество заключено в возможности визуализации произвольных последовательностей аргументов с помощью лишь небольшого числа примитивов и компактной базовой логики. Реализации, основанные на конструкциях списков аргументов, часто ограничены традиционной экспрессивностью первого порядка и не поддерживают все выводы, включающие квалификаторы по спискам. Это можно рассматривать как преимущество или как недостаток.

6.6 Особые случаи и переводы между диалектами

Диалект, в котором все операторы и предикаты являются недискурсивными именами, а все недискурсивные имена являются операторами или предикатами, называют классическим.

Интерпретацию I относят *к* отдельной вселенной, если $UD_I = UR_I$. Интерпретация I является экстенсиональной, когда rel_I b fun_I представляют собой функцию идентичности в (UR_I - UD_I), а сущности в исходной вселенной вне домена являются расширениями недискурсивных имен. В классических диалектах необходимо рассматривать только экстенсиональные интерпретации: для любой данной интерпретации I существует экстенсиональная интерпретация J, которая удовлетворяет тем же выражениям любого текста диалекта, что и I. J можно получить, заменив I(x) на $fun_I(I(x))$ для каждого оператора, заменив x на $rel_I(I(x))$ для каждого предиката x в словаре и удалив их из предметной области, если таковые есть. Поскольку все операторы и предикаты в классическом диалекте влияют на условия истинности только через ассоциированные расширения, это не затронет никаких значений истинности. Формально, $UD_J = UD_I$ - {I(v): v — оператор или предикат в V}, $int_J(x) = int_I(x)$ для дискурсных имен, $int_J(x) = rel_I(int_I(x))$ для предикатов x и $int_J(x) = fun_I(int_I(x))$ для операторов x.

Рекомендации по переводам между диалектами приведены в приложении Г.

7 Совместимость

7.1 Совместимость диалектов

7.1.1 Синтаксис

Диалект определяют на основе определенного набора надписей, который необходимо указать. Обычно в его качестве выступают строки символов Юникода (как указано в [6]), но возможно и использование других надписей, например схематических представлений, таких как ориентированные графы и структурированные изображения. Для диалекта необходимо указать метод, который будет однозначно анализировать все надписи в наборе или отклонять их как синтаксически недопустимые. Для надписей в строках символов Юникода грамматика в EBNF является достаточно точным определением. Анализ — это присвоение каждой части допустимой надписи соответствующей категории абстрактного синтаксиса CL, описанной в 6.1.1. Грамматически разобранная надпись является выражением.

Диалект, который содержит только некоторые типы выражений CL, считают синтаксически неполным диалектом CL или синтаксически частично совместимым диалектом. В частности, диалект, который не включает маркеры последовательности, но в остальном является полностью совместимым, известен как синтаксически компактный диалект. Описание некоторых отношений между синтаксической и семантической совместимостью см. в 7.1.

Диалект является синтаксически полностью совместимым, если в ходе его анализа есть возможность распознать выражения для каждой категории абстрактного синтаксиса из 6.1.1. Для обеспечения совместимости с CL диалекты или субдиалекты, анализ которых включает другие категории высказываний, должны либо (а) классифицировать их как нерегулярные высказывания, либо (б) определять, как эти категории сопоставляются с абстрактными синтаксическими категориями, определенными в 6.1.1. Если диалект удовлетворяет условию (а), такой диалект или субдиалект следует называть семантическим расширением (см. 6.1.2). Он является совместимым как синтаксический субдиалект, если признает хотя бы одну из категорий CL. При этом любой диалект должен признавать определенную форму категории высказывания. Выделяют три частных случая синтаксического субдиалекта. Компактный субдиалект — это диалект, который не признает маркеры последовательности. Инструктированный субдиалект — это диалект, который не распознает заголовки и операторы импорта. Субдиалект одной предметной области — это диалект, который не распознает ограничения предметной области.

Диалект является синтаксически сегрегированным, если в ходе его анализа необходимо различать лексические категории имен CL для проверки допустимости выражения на этом диалекте. Сегрегированные диалекты должны содержать критерии, достаточные для выявления приложением категории имени в диалекте без выполнения каких-либо операций с другими структурами помимо имени.

7.1.2 Семантика

Любой диалект CL должен иметь теоретико-модельную семантику, определенную на примере множества интерпретаций, называемых диалектными интерпретациями. Такая семантика присваивает одно из двух значений истинности — *true* или *false* — каждому утверждению, высказыванию или тексту на этом диалекте.

Диалект является семантически слабо совместимым, если для любого синтаксически допустимого предложения (кроме комментария) или текста T на этом диалекте существует сопоставление *tr* между выражениями на диалекте и выражениями с абстрактным синтаксисом CL, сопоставление *mod* между интерпретациями CL и диалектными интерпретациями, а также выполняется следующее условие совместимости:

- *mod*(I) подразумевает T, только если I подразумевает *tr*(T) для каждой интерпретации CL I *tr*(T).

Диалект является семантически полностью совместимым, если для любого синтаксически допустимого высказывания, утверждения (кроме комментария) или текста T на этом диалекте существует сопоставление *tr* между выражениями на диалекте и выражениями с абстрактным синтаксисом CL, а также выполняется следующее условие совместимости:

- текст T’ подразумевается T, только если *tr*(T’) подразумевается *tr*(T).

Диалект является широко совместимым в случае его слабой семантической совместимости, и при этом *mod* является сюръективным сопоставлением.

Диалект является семантически совместимым с субъязыком, если для любого синтаксически допустимого высказывания, утверждения (кроме комментария) или текста T на этом диалекте существует сопоставление *tr* между выражениями на диалекте и выражения с абстрактным синтаксисом CL, сопо-

ставление *mod* между интерпретациями CL и интерпретациями диалекта, а также выполняются следующие условия совместимости:

- диалект является семантически слабо совместимым;
- *tr* является инъективным сопоставлением;
- *mod* является биективным сопоставлением.

Диалект является семантически точно совместимым, если для любого синтаксически допустимого высказывания, утверждения (кроме комментария) или текста T на этом диалекте существует сопоставление *tr* между выражениями на диалекте и выражениями с абстрактным синтаксисом CL, сопоставление *mod* между интерпретациями CL и интерпретациями диалекта, а также выполняются следующие условия совместимости:

- диалект является семантически слабо совместимым;
- *tr* является биективным сопоставлением;
- *mod* является биективным сопоставлением.

Отсюда следует, что понятия выполнимости, противоречивости и следования, соответствующие интерпретациям диалекта и интерпретациям CL, идентичны для точно совместимого диалекта.

Самый простой способ достичь точной семантической совместимости — использовать теорию моделей CL в качестве теоретико-модельной семантики для диалекта, сформулировав при этом определение таким образом, чтобы дать возможность использовать другие способы формулировки семантической метатеории, если таковые предпочтительны по математическим или другим причинам, при условии сохранения выполнимости, противоречивости и следования.

Семантический субдиалект — это синтаксический субдиалект (см. 7.1.1), который соответствует семантическим условиям, указанным в таблицах 1 и 2. Он признает только некоторые части полной общей логики, а его интерпретации эквивалентны ограничениям интерпретации CL для этих частей.

Семантическое расширение — это диалект, который удовлетворяет первому условию, но не удовлетворяет второму. Другими словами, диалект семантического расширения имеет некоторую(ые) часть(и), чья интерпретация более ограничена, чем интерпретация CL. Любой диалект, который содержит нетривиальные семантические условия для нерегулярных высказываний, является семантическим расширением в этом смысле.

Это позволяет семантическому расширению налагать «внешние» семантические условия на нерегулярные высказывания в дополнение к семантическим условиям CL. На основании семантических условий, которые оно накладывает на числа и строки в кавычках, CLIF можно считать примером семантического расширения.

Семантические расширения следует обозначать как «совместимое семантическое расширение» или «совместимое расширение», а не как «точно совместимое» или просто как «совместимое». Для высказывания, утверждений и текстов совместимого расширения, противоречие и следование по отношению к семантике CL подразумевают, соответственно, противоречие и следование по отношению к семантике диалекта, но не наоборот. Удовлетворенность в отношении семантики диалекта предполагает удовлетворенность в отношении семантики CL, но не наоборот. Это означает, что в диалекте машины логического вывода, которые генерируют выводы для CL, будут работать правильно, но могут быть неполными.

Диалекты не могут ограничивать диапазон кванторов других диалектов. В других диалектах все имена могут быть рассмотрены как дискурсивные.

7.1.3 Пресуппозиционные диалекты

В диалектах CL может требоваться частичная или полная пресуппозиция дискурса в качестве режима следования для текстов. Характеристики пресуппозиции дискурса должны быть однозначно указаны для каждого текста на диалекте, но в остальном ее условия остаются произвольными, например: пресуппозиция может быть основана на соглашении об именах или получена путем использования имен в тексте.

Традиционная логика первого порядка как диалекта языка CL является пресуппозиционной. Все имена, используемые в качестве функциональных операторов или предикатов, описывает дискурсивная пресуппозиция «недискурса», а все имена, используемые в качестве аргументов функциональных терминов или простых высказываний или в качестве обязательных условий, описывает пресуппозиция «дискурса».

Диалекты CL, имеющие отношение к отдельной вселенной, также являются пресуппозиционными, причем дискурсивная пресуппозиция «дискурса» описывает все имена.

Любой диалект CL может включать синтаксическую конструкцию для использования внешней пресуппозиции дискурса в качестве предполагаемого режима следования текста.

7.2 Совместимость приложения

«Приложение» — это любая часть вычислительного оборудования (программного обеспечения, оборудования или сети), которая выполняет любые операции с текстом CL (даже очень тривиальные, например его сохранение для последующей повторной передачи).

Совместимость приложений определяют относительно набора диалектов, называемого набором совместимости. Приложения, которые совместимы с диалектом XCL, можно назвать «совместимыми» без каких-либо оговорок.

Все совместимые приложения должны быть способны обрабатывать все допустимые надписи диалектов в наборе совместимости. Приложения, которые вводят, выводят или передают текст CL, даже если они встроены в текст, обрабатываемый с использованием других текстовых соглашений, должны быть способны обрабатывать любой текст CL. Они должны выводить или передавать точные копии введенной надписи без искажения текста.

Приложения, которые обнаруживают следственные отношения между текстами CL в наборе соответствия, являются правильными, если при обнаружении приложением следования T из S в любых текстах T и S на диалектах в наборе совместимости такое S, в соответствии с CL, подразумевает T [то есть для любой интерпретации I в CL, если I(S) = true, то I(T) = true]. Приложение считают полным, если при обнаружении приложением следования T из S в соответствии с CL в любых текстах T и S на диалектах в наборе совместимости приложение может обнаружить следование T из S.

Примечание — Для этого необходима полнота «поперечных» диалектов в наборе совместимости.

Полнота не требует, чтобы приложение могло обнаруживать следование в семантическом расширении, которое не является следованием в соответствии с CL. Если диалект является семантическим расширением, тогда приложение является полным диалектом для этого диалекта, если для любой диалектной интерпретации I этого диалекта I (T) = true, если (S) = true, а приложение обнаруживает следование T из S.

7.3 Совместимость сетей

Совместимость сетей передачи данных определяют относительно набора диалектов, называемого набором совместимости. Сеть является совместимой, если она без искажений передает все выражения всех диалектов в наборе совместимости между любыми узлами сети и предоставляет сетевые идентификаторы, которые соответствуют семантическим условиям E17, E20 и требованиям 6.2. Ошибки или сбои передачи данных в сети, которые обозначены как состояния ошибок, не считают искажением при определении совместимости сети.

Приложение А
(обязательное)

Формат обмена общей логикой (CLIF)

А.1 Общие сведения

Исторически проект CL возник в результате попытки обновить и рационализировать язык KIF [1], который был впервые предложен в качестве «формата обмена знаниями» более десяти лет назад и в упрощенной форме *де-факто* стал стандартной системой обозначений во многих сферах применения логики. Некоторые особенности CL, в первую очередь использование маркеров последовательности, явным образом заимствованы из KIF. Однако концепция CL имеет определенные отличия от KIF, которые в сжатой форме рассматриваются в настоящем стандарте.

Во-первых, эти языки преследуют разные цели. Язык KIF был задуман как общее обозначение для перевода с множества других языков без потери смысла. CL предназначен для обмена информацией через сеть без выполнения какого-либо перевода (по возможности). При выполнении перевода CL предоставляет единую общую семантическую структуру, а не синтаксически определенный промежуточный язык.

Во-вторых, в значительной степени в результате вышесказанного, KIF рассматривался как «полный» язык, содержащий репрезентативный синтаксис для широкого спектра форм выражений, включая, например, сортировку кванторов, различные форматы определений и полностью экспрессивный метаязык. KIF должен был стать единым языком, с помощью которого можно было бы выразить множество других языков. Языку CL, напротив, специально отводилась роль «малого» языка, что позволяет упростить определение точной семантики и наложение точных границ экспрессивности подмножеств языка, а также позволяет определять расширенные языки в качестве кодировки аксиоматических теорий, выраженных в CL.

В-третьих, в основе KIF лежит язык программирования LISP. Синтаксис KIF определен как S-выражения языка LISP, а основанные на LISP идеи были включены в семантику KIF, например в виде определения семантики переменных последовательности. Хотя поверхностный синтаксис CLIF внешне напоминает LISP за счет использования вложенных немаркированных круглых скобок и может быть легко проанализирован в качестве S-выражений LISP, CL не основан на LISP и не содержит никаких допущений относительно каких-либо структур LISP. Рекомендуемая нотация взаимозамены CL основана на XML — стандарте, который не существовал во время первоначальной разработки KIF.

Наконец, многие из «новых» функций CL были непосредственно основаны на идеях, вытекающих из новой работы о языках для семантической сети [7].

Основанный на KIF синтаксис в CL называют CL Interchange Format (CLIF). Это упоминание приведено в связи с тем, что эта версия формата рекомендована настоящим стандартом, а также чтобы выделить ее на фоне других диалектов KIF, которые могут не быть полностью совместимыми.

Форматы KIF и CLIF во многом похожи. В качестве субдиалектов оба языка содержат синтаксис классической логики первого порядка (FO). Оба языка имеют обозначения для переменных последовательности (в настоящем стандарте называемых маркерами последовательности). Оба языка следуют исключительно префиксным условным обозначениям и соглашениям о синтаксисе стиля для S-выражений. Оба используют круглые скобки в качестве лексических разделителей. Оба одинаковым образом определяют ограничения квантора.

Ниже описаны некоторые известные различия между KIF и CLIF:

а) KIF использует кодировку ASCII, а CLIF — Юникод;

б) в KIF, в отличие от CLIF, есть явные обозначения для определения функций и отношений;

в) в KIF не используют обозначение выделенных имен, доступное в CLIF;

г) в KIF в качестве префикса переменной последовательности используют символ «@», в CLIF маркеры последовательности обозначают тремя точками;

д) в KIF комментарии обрабатывают иначе, чем в CLIF, и отсутствует «изолирующая» конструкция;

е) в KIF, в отличие от CLIF, отсутствуют конструкции пар ролей;

ж) в KIF отсутствует тип нерегулярного высказывания, который в CLIF допускается для расширений языка;

и) в KIF отсутствуют понятия импорта, текстов, заявлений и ограничений области, которые есть в CLIF;

к) в KIF переменные имеют отличия от имен, а кванторы могут связывать только переменные. В CLIF такие различия отсутствуют;

л) свободные переменные в KIF обрабатывают как универсальные. Свободные имена в CLIF — это просто имена, квантификация которых не выполняется;

м) в KIF операторы и предикаты могут быть только именами. В CLIF допускают использование общих терминов, а имена могут быть связаны кванторами;

н) в KIF не поддерживается конструкция защищенного квантора.

А.2 Синтаксис CLIF

А.2.1 Символы

В соответствии с [6] все выражения CLIF кодируют в качестве последовательности символов Юникода. Может быть использована любая кодировка символов, которая соответствует требованиям [6], но не следует использо-

вать UTF-8 (см. [6], приложение Г). Чтобы при необходимости иметь возможность закодировать язык в виде текстовой строки ASCII, в CLIF отдельно зарезервированы символы из подмножества US-ASCII. В настоящем стандарте используют символы ASCII. Символы Юникода за пределами диапазона ASCII представлены в тексте CLIF ASCII в виде кодирующей последовательности символов \u*nnnn* или \U*nnnnnn*, где *n* — шестнадцатеричный цифровой символ. При преобразовании текстовой строки ASCII в полнозначную кодировку символов, а также при печати или ином отображении текста для обеспечения максимальной читаемости такую последовательность можно заменить соответствующей прямой кодировкой символа или соответствующего глифа. Более того, если такие кодирующие последовательности встречают в строках в кавычках (см. ниже), считают, что они обозначают соответствующий символ Юникода.

Синтаксис определяют в терминах непересекающихся блоков символов, называемых лексическими токенами (в соответствии ГОСТ 33707 о лексических токенах). Поток символов можно преобразовать в поток лексических токенов с помощью простого процесса лексикализации, в ходе которого идет проверка наличия небольшого количества символов-разделителей, которые указывают на завершение одного лексического токена и, возможно, начало следующего лексического токена. Последовательности пробелов выступают в качестве разделителей между лексическими токенами (кроме строк и имен в кавычках; см. ниже). Некоторые символы зарезервированы для использования в качестве первого символа в лексической единице. Символ двойной кавычки (U+0022) принято ставить в начале и конце имен, которые содержат символы-разделители, символ одинарной кавычки (апостроф, U+002C) принято ставить в начале и конце строк в кавычках, которые также являются лексическими единицами, способными содержать символы-разделители, а знак равенства, если он стоит в самом начале лексической единицы, должен выступать в качестве отдельной единицы.

Для символа обратной косой черты \ (U+005C) зарезервирована отдельная функция. Если за ним следует буква u или U и четырех- или шестизначный шестнадцатеричный код, соответственно, его используют для расшифровки символов Юникода, которые не относятся к ASCII, в потоке символов ASCII, как описано выше. При обработке строки ASCII любая строка такой формы играет в CL ту же синтаксическую роль, что и отдельный обычный символ. Комбинацию \' (U+005C, U+002C) используют для кодирования одинарной кавычки внутри заключенной в кавычки строки CL, а комбинация \" (U+005C, U+0022) обозначает двойную кавычку в строке выделенного имени, заключенной в двойные кавычки. В обоих случаях обратную косую черту обозначают двумя обратными косыми чертами \\ (U+005C, U+005C). Любое другое применение обратной косой черты является ошибкой. Эти внутренние соглашения о кавычках применяют при обработке строк как в кодировке ASCII, так и в Юникоде.

А.2.2 Лексический синтаксис

А.2.2.1 Общие положения

Для удобства разделения презентации на две части различают лексические и синтаксические конструкции. Этот подпункт может помочь разработчикам в определении логических маркеров, составляющих синтаксические выражения, как показано в А.2.3. Данные характеристики не являются обязательными при реализации.

А.2.2.2 Пробел

```
white = space U+0032 | tab U+0009 | line U+0010 | page U+0012 | return U+0013;
```

А.2.2.3 Разделители

Одиночные кавычки (апостроф) используют для разделения строк в кавычках, а двойные кавычки — для разделения выделенных имен, которые следуют специальным правилам лексикализации. Строки в кавычках и выделенные имена — единственные лексические элементы CLIF, которые могут содержать пробелы и круглые скобки. Круглые скобки в других местах являются самоограничивающими, их считают самостоятельными лексическими лексемами. Круглые скобки — это основное средство группировки в синтаксисе CLIF.

```
open = '(' ;

close = ')' ;

stringquote = "'" ;

namequote= '"' ;

backslash= '\' ;
```

А.2.2.4 Символы

char — это все оставшиеся неуправляющие символы ASCII, которые можно использовать для формирования лексических токенов (с учетом некоторых ограничений, связанных с первым символом лексического токена). Сюда входят все буквенно-цифровые символы.

```
char = digit | '~' | '!' | '#' | '$' | '%' | '^' | '&' | '*' | '_' | '+' | '{' | '}' | '|' |
    ':' |    '<' | '>' | '?' | '`' | '-' | '=' | '[' | ']' |   ';'| ',' | '.' | '/' | 'A' |
    'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' |
    'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | 'a' | 'b' | 'c' |
    'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' |
    'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' ;

digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' ;

hexa = digit | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' ;
```

A.2.2.5 Использование кавычек внутри строк

Определенные последовательности символов используют для обозначения наличия отдельного символа. *nonascii* — это набор символов или последовательностей символов, который указывает на символ Юникода за пределами диапазона ASCII.

При вводе данных с помощью полной кодировки символов Юникода это определение следует игнорировать. nonascii следует рассматривать как набор всех неуправляющих символов Юникода за пределами диапазона ASCII, поддерживаемых кодировкой символов. Использовать последовательности \uxxxx и \Uxxxxxx в тексте, закодированном с помощью полного набора символов Юникода, не рекомендуется.

innerstringquote используют для обозначения наличия символа одинарной кавычки внутри строки в кавычках. Строка в кавычках может содержать любой символ, включая пробелы. Тем не менее символ одинарной кавычки может встречаться внутри строки, заключенной в кавычки, только как часть *innerstringquote*, т. е. когда прямо перед ним стоит символ обратной косой черты. Символ одинарной кавычки в потоке символов строки в кавычках обозначает конец лексического токена строки в кавычках, если прямо перед ним не стоит символ обратной косой черты. Двойные кавычки внутри строк выделенных имен обрабатывают таким же образом. *Innernamequote* используют для обозначения наличия символа двойной кавычки внутри выделенного имени.

```
nonascii = '\u' , hexa, hexa, hexa, hexa | '\U' , hexa, hexa, hexa, hexa, hexa, hexa ;

innerstringquote = '\'' ;

innernamequote = '\"' ;

innerbackslash= '\\'

numeral = digit , { digit } ;
```

Маркеры последовательности — это отдельная синтаксическая форма в CL со специальным значением. Обратите внимание, что многоточие без текста (например, **«...»**) само по себе является маркером последовательности.

```
seqmark = '...' , { char } ;
```

Одиночные кавычки являются разделителями для строк в кавычках; двойные кавычки — для выделенных имен. Выделенное имя — это простое имя, которое может содержать символы, нарушающие лексикализацию, например «Миссис Нора Джонс» или «Девушка(прервано)»; как и любое другое имя, оно может обозначать что угодно. Окружающие двойные кавычки не считаются частью имени, которое определяют как строку символов, полученную путем удаления двойных кавычек и замены всех внутренних экземпляров innernamequote на одинарный символ двойной кавычки. При использовании цифр в двойных кавычках во время синтаксического анализа абстрактного синтаксиса этому интерпретируемому имени присваивают новый символ. Рекомендовано использовать синтаксис выделенного имени при использовании URI, ссылок URI и IRI в качестве имен, поскольку эти веб-идентификаторы могут содержать символы, которые в противном случае нарушили бы лексикализацию CLIF: в частности, совместимые с Xpath ссылки URI часто заканчиваются закрывающими скобками.

Строка в кавычках, напротив, является выражением с фиксированным семантическим значением: она *обозначает* текстовую строку, аналогичную строке внутри кавычек.

А.2.2.6 Строки в кавычках

Строки в кавычках и выделенные имена требуют применения другого алгоритма лексикализации, чем остальные части текста CLIF, поскольку круглые скобки и пробелы не разбивают поток текста в кавычках на лексические токены.

Если текст CLIF заключен в текст или документ, опирающийся на соглашения об экранировании символов, подразумевают, что описанные здесь соглашения о строках CL в кавычках могут быть применены к тексту, описанному или обозначенному на основании используемых соглашений, которые подлежат применению в первую очередь. Таким образом, например, содержимое элемента XML

<cl- text>'a\'b<c&apos</cl-text> является строкой в кавычках с синтаксисом CLIF ‘a\’b<c’, которая обозначает пятисимвольную текстовую строку a’b<c. Тем не менее, простой текст CLIF 'a\'b<c&apos представлял бы собой довольно длинное имя.

---

quotedstring = stringquote, { white | open | close | char | nonascii | namequote |

innerstringquote | innerbackslash }, stringquote ;

enclosedname = namequote, { white | open | close | char | nonascii | stringquote |

innernamequote }, namequote

А.2.2.7 Зарезервированные токены

reservedelement состоит из лексических токенов, которые используют для обозначения синтаксической структуры выражений CL. Их не допускается использовать в качестве имен в тексте CLIF.

---

reservedelement = ‘=’ | ‘and’ | ‘or’ | ‘iff’ | ‘if’ | ‘forall’ | ‘exists’ |

‘not’ | ‘cl:text’ | ‘cl:ttl’ | ‘cl:imports’ | ‘cl:restrict’ | ‘cl:indiscourse’ |

‘cl:outdiscourse’ | ‘cl:comment’ | ‘cl:prefx’ ;

А.2.2.8 Последовательность символов имени

namecharsequence — это лексический токен, который не начинается с каких-либо специальных символов. Обратите внимание, что namecharsequences может не содержать пробелов или круглых скобок, а также может не начинаться с кавычек, хотя и может их содержать. Цифры и маркеры последовательностей не являются namecharsequence.

---

namecharsequence = (char , { char | stringquote | namequote | backslash }) - (

reservedelement | numeral | seqmark) ;

А.2.2.9 Лексические категории

Задача лексического анализатора — разобрать поток символов на последовательные, непересекающиеся строки lexbreak и nonlexbreak и передать выявленные лексические токены в виде потока токенов на следующий этап синтаксической обработки. Лексические лексемы делят на восемь взаимно непересекающихся категорий: открывающие и закрывающие круглые скобки, числа, строки в кавычках (которые начинаются с и заканчиваются на “’”), маркеры последовательности (которые начинаются с ‘...’), выделенные имена (которые начинаются с и оканчиваются на “”), а также namesequence и зарезервированные элементы.

---

lexbreak = open | close | white , { white } ;

nonlexbreak = numeral | quotedstring | seqmark | reservedelement | namecharsequence |

enclosedname ;

lexicaltoken = open | close | nonlexbreak ;

charstream = { white } , { lexicaltoken, lexbreak } ;

А.2.3 Синтаксис выражений

А.2.3.1 Последовательность терминов

Термы и атомарные высказывания используют понятие последовательности терминов, представляющее вектор аргументов к функции или отношению. Маркеры последовательности используют для обозначения подпоследовательности последовательности терминов; термины обозначают отдельные элементы.

А.2.3.2 Имя

Имя — это любой лексический токен, который рассматривают как обозначение элемента. Имена, которые имеют фиксированное значение, отличаются от имен, значение которых определяется интерпретацией.

```
termseq = { term | seqmark } ;

cseqmark = seqmark | ( open, 'cl:comment', quotedstring , seqmark , close ) ;
```

```
interpretedname = numeral | quotedstring | ( open, 'cl:comment', quotedstring , (numeral |
        quotedstring) , close ) ;

interpretablename = namecharsequence | enclosedname |( open, 'cl:comment', quotedstring ,
interpretablename , close );

name = interpretedname | interpretablename ;
```

А.2.3.3 Термин

Имена считаются терминами, а функциональный термин состоит из оператора, который сам по себе является термином, и вектора аргументов. С терминами также могут быть связаны комментарии, представленные в виде строки в кавычках (это необходимо для использования текста, который в противном случае нарушил бы лексикализацию). Оболочки комментариев синтаксически выделяют термин, который комментируют.

```
term = name | ( open, operator, termseq, close ) | ( open, 'cl:comment', quotedstring ,
        term, close ) ;

operator = term ;
```

А.2.3.4 Уравнение

Уравнения выделены в особую категорию из-за своей особой семантической роли и особой процедуры обработки во многих приложениях. Знак равенства не является именем.

```
equation = open, '=', term, term, close ;
```

А.2.3.5 Высказывание

Как и термины, высказывания могут иметь выделяющие комментарии. Обратите внимание, что комментарии могут относиться к высказываниям, которые являются частями более крупных высказываний.

```
sentence = atomsent | boolsent | quantsent | commentsent ;
```

А.2.3.6 Атомарная последовательность

Атомарные высказывания похожи по структуре на термины, но их аргументы могут быть представлены с помощью пар ролей, состоящих из имени роли и термина. Уравнения считаются атомарными высказываниями, а атомарное высказывание может быть представлено с помощью ролевых пар, состоящих из имени роли и термина.

```
atomsent = equation | atom ;

simple_sentence = ( open, predicate , termseq, close ) ;

predicate = term ;
```

А.2.3.7 Логическое высказывание

Логические высказывания требуют, чтобы импликация и двойная импликация были двоичными, но разрешают конъюнкции и дизъюнкции иметь любое количество аргументов, включая ноль. Высказывания (and) и (or) могут использоваться как значения true и false соответственно.

```
boolsent = ( open, (‘and’ | ‘or’) , { sentence }, close ) | ( open, (‘if’ | ‘iff’) ,
sentence , sentence, close ) | ( open, ‘not’ , sentence, close ;
```

А.2.3.8 Количественное высказывание

Кванторы могут связывать любое количество переменных, а связанные переменные могут быть ограничены категорией, указанной термином.

```
quantsent = open, (‘forall’ | ‘exists’) , boundlist, sentence, close ;

boundlist = open, bvar, { bvar } , close ;

bvar = interpretablename | cseqmark

(open, (interpretablename | cseqmark), term, close ) ;
```

А.2.3.9 Прокомментированное высказывание

Комментарий может иметь отношение к любому предложению, поэтому комментарии можно прикрепить к высказываниям, которые являются частями выражений более крупных высказываний.

```
commentsent = open, ‘cl:comment’, quotedstring , sentence , close ;
```

А.2.3.10 Заголовок

Заголовок CLIF содержит имя текста.

```
titling = open, ‘cl:ttl’, interpretable name , text , close ;
```

А.2.3.11 Дискурсивное утверждение

Дискурсивное утверждение CLIF — это либо утверждение в дискурсе (которое определяет набор терминов, обозначающих элементы во вселенной дискурса), либо утверждение вне дискурса (которое определяет набор терминов, которые не обозначают элементы во вселенной дискурса).

```
indiscourse = open, ‘cl:indiscourse’, term, {term} , close ;

outdiscourse = open, ‘cl:outdiscourse’, term, {term} , close ;

discoursestatement = indiscourse | outdiscourse ;
```

А.2.3.12 Утверждение

Утверждение CLIF — это либо заголовок либо дискурсивное высказывание (возможно, с комментарием).

```
statement = titling | discoursestatement | ( open, ‘cl:comment’, quotedstring , statement ,
close ) ;
```

А.2.3.13 Импорт

Импорт CLIF содержит заголовок, который служит идентификатором для внешнего текста CL.

```
importation = open, ‘cl:imports’, interpretablename , close ;
```

А.2.3.14 Ограничение области

Ограничения области — это именованные текстовые сегменты, которые представляют текст, предназначенный для понимания в «локальном» контексте, где имя указывает на область кванторов в тексте. Имя текста не может быть числом или строкой в кавычках. Обратите внимание, что текст и ограничение области являются взаимно рекурсивными категориями, поэтому ограничения области могут быть вложенными.

```
domainrestriction = open, ‘cl:restrict , term , text, close ;
```

A.2.3.15 Текст
Текст CLIF — это текстовая конструкция, импорт или ограничение области.

```
textconstruction = open, 'cl:text', { sentence | statement | text }, close ;

prefxdeclaration = open, 'cl:prefx', (quotedstring - 'cl'), interpretablename, close ;

commenttext = open, 'cl-comment', quotedstring, {prefxdeclaration}, cltext, close ;

text = textconstruction | domainrestriction | importation | commenttext ;

cltext = {text} ;
```

A.3 Семантика CLIF

Семантика CLIF аналогична семантике абстрактного синтаксиса CL, описанной в 6.2.

Примечание — Интерпретация любого выражения CLIF определяется записями из таблицы 1. Обозначение <T_1 … T_n> указывает на последовательность терминов в терминах синтаксиса и на последовательность, то есть элемент U_I*, в терминах семантики. В первом столбце приведены ссылки на строки из таблицы 2.

Таблица А.1 — Семантика CLIF

	Если E является выражением в виде	Тогда I(E) =
E1	Десятичное число	Натуральное число, обозначаемое десятичной цифрой
E1	Строка в кавычках 's'	Строка символов Юникода, образованная удалением внешних одинарных кавычек и заменой экранированных внутренних подстрок на эквиваленты из Юникода
E1, E2	Интерпретируемое имя	I(E) = int_I(E)
E3	Последовательность терминов <T_1 … T_n>, которая начинается с термина T_1	I(E) = <I(T_1)>;I(<T_2 … T_n>]
E4	Последовательность терминов T_1 … T_n, которая начинается с маркера последовательности T_1	I(E) = I(T_1); I(<T_2 … T_n>]
E5	Термин (O T_1 … T_n)	I(E) =fun_I (I(O))(I(<T_1 … T_n >]
	Имя, термин или маркер последовательности (CL: откомментированная «строка» T)	I(E) = I(T)
E6	Уравнение (= T_1 T_2)	I(E) = true, если I(T_1) = I(T_2); в противном случае I(E) = false
E7	Атомарное высказывание (P T_1 … T_n)	I(E) = true, если I(<T_1 … T_n>) находится в rel_I(I(P)); в противном случае I(E) = false
E8	Логическое высказывание (not P)	I(E) = true, если I(P) = false; в противном случае I(E) = false
E9	Логическое высказывание (and P_1 … P_n)	I(E) = true, если I(P_1) =…I(P_n) = true; в противном случае I(E)= false
E10	Логическое высказывание (or P_1 … P_n)	I(E) = false, если I(P_1) = …I(P_n) = false; в противном случае I(E) = true
E11	Логическое высказывание (if P Q)	I(E) = false, если I(P) = true и I(Q) = false; в противном случае I(E) = true

Окончание таблицы А.1

	Если E является выражением в виде	Тогда I(E) =
E12	Логическое высказывание (iff P Q)	I(E) = true, если I(P) = I(Q); в противном случае I(E) = false
	Высказывание или утверждение (cl: comment “string” P)	I(E) = I(P)
E13	Количественное высказывание (forall (N_1 … N_n) B), где N = {N_1, …, N_n} является набором обязательных условий для высказывания	I(E) = true, если для каждой N-версии J в I, J(B) = true; в противном случае I(E) = false
E14	Количественное высказывание (exists (N_1 … N_n) B), где N = {N_1, …, N_n} является набором обязательных условий для высказывания	I(E) = true, если для части N-версий J в I, J(B) является истинным; в противном случае I(E) = false
	Правильно построенное выражение (cl: comment “string”)	I(E) = true
E17	Импорт (cl: imports N)	I(E) = true
E19	Текстовая конструкция (cl: text T_1 … T_n)	I(E) = true, если I(T_1)= … = I(T_n) = true; в противном случае I(E) = false
E20	Заголовок (cl: ttl N T)	I(E) = true, если ttl_I(N) = T; в противном случае I(E) = false
	Если E является выражением в виде	Тогда *I(E)* =
E21	Утверждение в дискурсе (cl: indiscourse T1 … Tn)	*I(E)* = true, если *I(Ti)* ∈ *UD/* ∪ *UD/** для *1* ≤ *i* ≤ *n;* в противном случае *I(E)* = false
E22	Утверждение вне дискурса (cl: outdiscourse T1 … Tn)	*I(E)* = true, если *I(Ti)* ∉ *UD/* ∪ *UD/** для *1* ≤ *i* ≤ *n;* в противном случае *I(E)* = false
E23	Ограничение области (cl: restricts N T)	*Текст T [T']),* где *T'* — это текст *T*, в котором каждое имя или маркер последовательности *X* в списке boundlist квантора заменяется на *(X N)*

Таблица А.1 не включает в себя все синтаксические формы CLIF. Интерпретация остальных синтаксических случаев зависит от их сопоставления с другими выражениями CLIF, интерпретация которых приведена в таблице А.1. Перевод определяется в таблице А.2, где указан перевод T [E] для выражения E.

Таблица А.2 — Сопоставление дополнительных форм CLIF с основными

Если E	То E переводится в T [E] =
Количественное высказывание (forall ((N_1 T_1) …) B)	Количественное высказывание (forall (N_1) T [(forall (…) (if (T_1 N_1) B)]
Количественное высказывание (exists ((N_1 T_1) …) B)	Количественное высказывание (exists (N_1) T [(exists (…) (and (T_1 N_1) B)]

Формы в левой части таблицы А.2 можно рассматривать как «синтаксический сахар» для своих переводов справа, которые, соответственно, называются их синтаксической солью, а субдиалект CLIF без этих выражений образует «соленый» CLIF.

А.4 Совместимость в CLIF

А.4.1 Синтаксическая совместимость

Соответствие синтаксиса CLIF абстрактному синтаксису CL обозначается с помощью обозначений в левом столбце таблицы А.1, которые идентичны обозначениям в таблице 2. С их помощью можно определить полное синтаксическое соответствие «соленого» CLIF в ходе проверки. Обратите внимание, что имена interpretedname и

interpretablename считаются именами CL. Синтаксическое соответствие CLIF следует особенностям сопоставления, определенным в таблицы А.2. Обратите внимание, что синтаксис комментариев CLIF обрабатывает выражения с комментариями как идентичные по значению выражениям без комментариев, поэтому комментарий можно считать «прикрепленным» к выражению без комментария.

А.4.2 Семантическая совместимость

CLIF — это семантическое расширение CL. Чтобы подтвердить, что CLIF является семантическим расширением CL, необходимо показать, что если I является интерпретацией CLIF, то должна существовать интерпретация CL, которая приписывает одинаковое значение истинности каждому высказыванию. Это будет продемонстрировано путем построения J из I с использованием обозначений и соглашений, приведенных выше при описании I и в 6.2 при описании J.

J имеет тот же словарь, что и I: $UD_J = UR_J = U_I$, $rel_J = rel_I$ и $fun_J = fun_I$. Интерпретация имен interpretablename выполняется очевидным образом: $int_J(x) = int_I(x)$ для любого interpretablename x. Поскольку имена interpretedname словаря CLIF классифицируются как имена CL, для $int_J(x)$ также необходимо давать определение, если x является interpretedname. Это делается для обеспечения соответствия первым двум записям в семантической таблице CLIF, то есть $int_J(x)$ = целое число, обозначаемое x, если x является десятичным числом, и $int_J(x)$ = строка символов Юникода, обозначаемая x, если x является строкой в кавычках CLIF. После этого сравнение случаев показывает, что J (s) = I (s) для любого высказывания s CLIF. Если s представляет собой текст с именем N, списком исключений L и основным текстом B, следует показать, что J(s) = true, только если [J<L](B) = true и rel(J(N)) = $UR_{[J<L]}$* (т. к. $UD_J = UR_J$). Нетрудно заметить, что это в точности эквивалентно истинности высказывания I в «соленом» переводе основного текста, соответствующем таблице А.2, как описано в 6.2. (При формальном доказательстве будет выполнена структурная индукция по высказываниям основного текста.) Следовательно, для любого текста t CLIF J(t) = I(t).

Особенность заключается не в том, что если I представляет собой любую интерпретацию CL текста t CLIF, то должна существовать интерпретация J CLIF, в которой t будет иметь такое же значение. Поскольку имена interpretedname CLIF обрабатываются в CL как обычные имена, J может присвоить им значение, которое несовместимо с их фиксированной интерпретацией в CLIF, например J('a string') = 3 не исключается правилами общей логической семантики. Это обычное явление для любого диалекта, которое связывает заранее определенные извне значения с некоторыми категориями имен, например числами или выражениями с типом данных. Такие диалекты могут поддерживать выводы, которые не допускается выразить как аксиомы CL, поэтому должны классифицироваться как внешние семантические расширения CL. Субдиалект CLIF, который не использует цифры и строки в кавычках, отличается полной семантической совместимостью, что можно показать, инвертировав приведенную выше конструкцию J из I.

Приложение Б
(обязательное)

Формат обмена концептуальными графами (CGIF)

Б.1 Общие положения

В этом разделе приведена краткая информация о концептуальных графах и описан набор правил преобразования (перезаписи), которые будут использоваться в остальной части этого приложения для определения описания синтаксических правил для CGIF.

Абстрактный синтаксис CG — это независимая от обозначений спецификация выражений и компонентов *ядра концептуального графа*, т. е. минимального подмножества CG, способного выражать полную семантику CL. Семантика любого выражения *x* в синтаксисе ядра CG задается функцией *cg2cl*(*x*), которая сопоставляет *x* с логически эквивалентным выражением с абстрактным синтаксисом CL. Функция *cg2cl* является рекурсивной, поскольку CG или ее компоненты можно вложить в другие компоненты.

В пунктах с Б.2.1 по Б.2.11 определяется абстрактный синтаксис CG, сопоставление абстрактного синтаксиса CG с абстрактным синтаксисом CL и соответствующий реальный синтаксис для ядра CGIF. Каждый подпункт включает формальное определение, сопоставление с CL, правило для реального синтаксиса CGIF и комментарий с объяснением и примерами. Правила синтаксиса записаны в расширенной форме Бэкуса — Наура (EBNF), как указано в [2], и кратко изложены в Б.1.3. Для каждого правила синтаксиса CGIF необходимо ввести лексические категории из п. А.2.2. В п. А.2.3.2 категория *имя* включает категорию *enclosedname* для строк в кавычках и категорию *namesequence* для строк без кавычек. Чтобы избежать возможных двусмысленностей, для категории *CGname* требуется, чтобы все последовательности имен CLIF, кроме указанных в идентификаторе категории CGIF, были заключены в кавычки:

CGname = identifer | '"', (namesequence - identifer), '"'
| numeral | enclosedname | quotedstring;
identifer = letter, {letter | digit | "_"};

При переводе CGIF в CL все имена CGname должны переводиться путем удаления любых кавычек вокруг последовательности имен. CLIF не подразумевает синтаксических различий между константами и переменными, но в CGIF любое имя CGname, которое не используется в качестве определяющей или связанной метки, должно называться константой.

Начальным символом для синтаксиса CGIF должен быть текст категории (если вводится полный текст) или категория CG (если вводится строка, представляющая концептуальный граф).

Б.1.1 Концептуальные графы

Концептуальный граф (CG) — это представление логики в виде двудольного графа с двумя типами узлов, которые называются концепциями и концептуальными отношениями. Формат обмена концептуальными графами (CGIF) — это полностью совместимый диалект CL (CL), который выступает в качестве упорядоченного представлением концептуальных графов. В данном приложении описывается синтаксис CGIF и выполняется его сопоставление с семантикой CL. Ненормативное графическое обозначение, называемое формой отображения CG, используется в настоящем стандарте только в примерах, которые иллюстрируют структуры CG. В первом примере на рисунке Б.1 показана форма отображения высказывания *John is going to Boston by bus* (Джон едет в Бостон на автобусе).

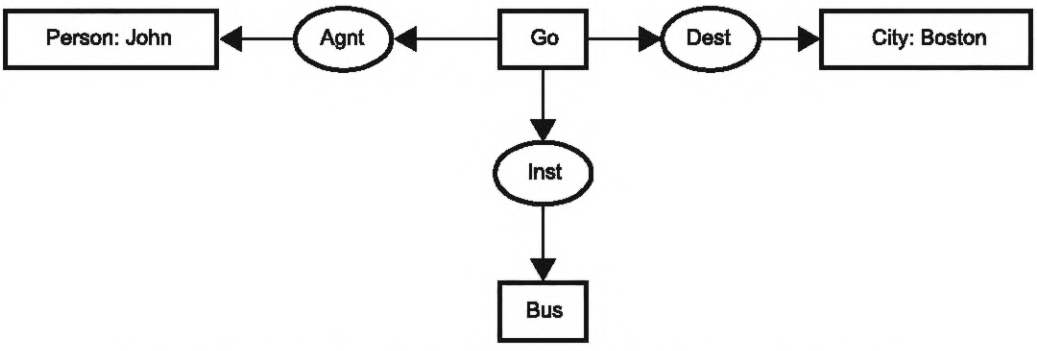

Рисунок Б.1 — Форма отображения CG «John is going to Boston by bus»

В форме отображения прямоугольники или поля представляют собой концепции, а круги или овалы — концептуальные отношения. Дуга со стрелкой, указывающей на круг, отмечает первый аргумент отношения, а дуга, указывающая в направлении от круга, — последний. Если отношение имеет только один аргумент, стрелка не используется. Если отношение имеет более двух аргументов, стрелки заменяются целыми числами 1,...,*n*.

CG на рисунке Б.1 имеет четыре концепции, каждая из которых имеет *метку типа*, представляющая собой тип объекта, к которому относится концепция: Person, Go, Boston и Bus. У двух концептов есть *константы*, которые идентифицируют объекты: John и Boston. Каждое из трех концептуальных отношений имеет метку типа, которая представляет тип отношения: Agnt — агент перемещения, Inst — инструмент, Dest — пункт назначения. CG в целом указывает на то, что John является агентом перемещения, Boston — пунктом назначения, а bus — инструментом. Ниже показано представление CGIF на рисунке Б.1:

[Go: *x] [Person: John] [City: Boston] [Bus: *y]

(Agnt ?x John) (Dest ?x Boston) (Inst ?x ?y)

В CGIF концепции представлены квадратными скобками, а концептуальные отношения — круглыми. Строка символов, перед которой стоит символ звездочки, например *x, является определяющей меткой, на которую можно ссылаться с помощью связанной метки ?x, перед которой стоит вопросительный знак. Эти строки, которые в CGIF называются метками кореферентности, соответствуют переменным в CLIF. Если перед определяющей меткой не стоит символ @every, она переводится в квантор существования. Ниже приведено эквивалентное представление CLIF рисунка Б.1:

(exists ((x Go) (y Bus))

(and (Person John) (city Boston)

(Agnt x John) (Dest x Boston) (Inst x y)))

Как показывает этот пример, различия между CGIF и CLIF связаны со структурой графа: узлы графа не имеют неявного упорядочивания, а метки кореферентности, такие как *x или ?x, обозначают соединения узлов, а не переменных. Обратите внимание, что CGIF использует префиксы * и ?, чтобы отличать метки кореферентности от констант, но в CLIF никаких синтаксических характеристик для различения переменных и констант не используется.

Рисунок Б.1 и его представление в CGIF иллюстрируют расширенный синтаксис CGIF, который добавляет метки типов к концепциям и несколько других синтаксических расширений к основному синтаксису. Для преобразования расширений расширенного синтаксиса в основной синтаксис CGIF метки типов в узлах концепций заменяются отношениями, связанными с этими узлами. Например, концепция [Go:*x] становится нетипизированной концепцией [*x] и концептуальным отношением (Go ?x). Концепция [Person: John] становится [:John] (Person John), что можно упростить до отношения (Person John). Ниже приведен основной синтаксис CGIF и соответствующий синтаксис CLIF:

```
[*x] [*y]
(Go ?x) (Person John) (City Boston) (Bus ?y)
(Agnt ?x John) (Dest ?x Boston) (Inst ?x ?y)

(exists (x y)
(and (Go x) (Person John) (City Boston) (Bus y) (Agnt x John) (Dest x Boston) (Inst x y)))
```

Для иллюстрации контекстов и логических операторов на рисунке Б.2 показана форма отображения высказывания *If a cat is on a mat, then it is a happy pet (Если кошка сидит на лежанке,* значит, *она* — счастливое домашнее животное*).* Как и на рисунке Б.1, прямоугольники представляют собой концептуальные узлы, но два больших прямоугольника содержат вложенные концептуальные графы. Любая концепция, содержащая вложенный CG, называется контекстом; в этом примере метки типа If и Then указывают, что предложение, сформулированное CG в контексте if, подразумевает предложение, сформулированное CG в контексте then. Отношение Attr указывает на то, что кошка, которую также называют домашним животным, имеет атрибут — счастье.

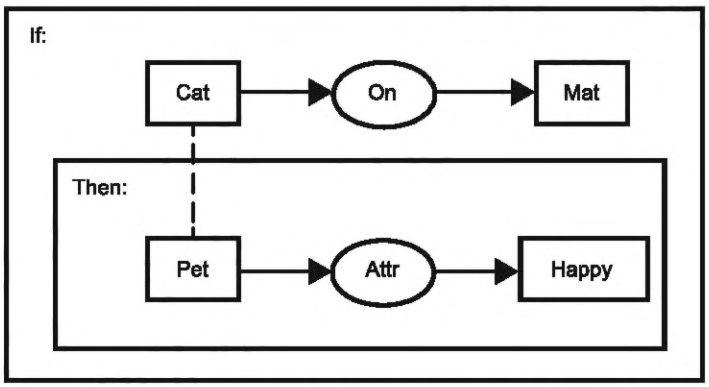

Рисунок Б.2 — Форма отображения CG для высказывания «If a cat is on a mat, then it is a happy pet»

Пунктирная линия, соединяющая понятия [Cat] и [Pet], является соединением кореферентности, которое указывает на то, что обе этих концепции относятся к одному и тому же объекту. В CGIF соединение показано определяющей меткой *x в концепции [Cat:*x] и связанной меткой ?x в концепции [Pet: ?x]:

[If: [Cat: *x] [Mat: *y] (On ?x ?y)

[Then: [Pet: ?x] [Happy: *z] (Attr ?x ?z)]]

В основном синтаксисе CGIF метки типов If и Then заменяются символом отрицания ~ перед открывающей скобкой, а метки типов — монадическими отношениями:

~[[*x] [*y] (Cat ?x) (Mat ?y) (On ?x ?y)

~[[*z] (Pet ?x) (Happy ?z) (Attr ?x ?z)]]

CLIF:

(not (exists (x y) (and (Cat x) (Mat y) (On x y)

(not (exists (z) (and (Pet x) (Happy z) (Attr x z)))))))

В основном синтаксисе CGIF единственный квантор — это квантор существования. В расширенном синтаксисе CGIF кванторы всеобщности могут использоваться для представления логически эквивалентного высказывания «For every cat and every mat, if the cat is on the mat, then it s a happy pet» («Для каждой кошки и каждой лежанки действует правило: если кошка находится на лежанке, это счастливое домашнее животное»). В расширенном синтаксисе CGIF квантор всеобщности представлен символом @every:

[Cat: @every *x] [Mat: @every *y]

[If: (On ?x ?y) [Then: [Pet: ?x] [Happy: *z] (Attr ?x ?z)]]

CLIF:

(forall ((x Cat) (y Mat))

(if (On x y) (and (Pet x) (exists ((z Happy)) (Attr x z)))))

В CG функции представлены концептуальными отношениями, называемыми действующими субъектами. На рисунке Б.3 показана форма отображения CG для следующего уравнения, записанного обычными алгебраическими символами:

y = (x + 7)/sqrt(7)

Три функции в этом уравнении будут представлены тремя действующими субъектами, которые показаны на рисунке Б.3 в виде узлов в форме ромба с метками типа Add, Sqrt и Divide. Поля представляют собой узлы концепций, которые содержат входные и выходные значения действующих субъектов. Две пустых концепции содержат выходные значения Add и Sqrt.

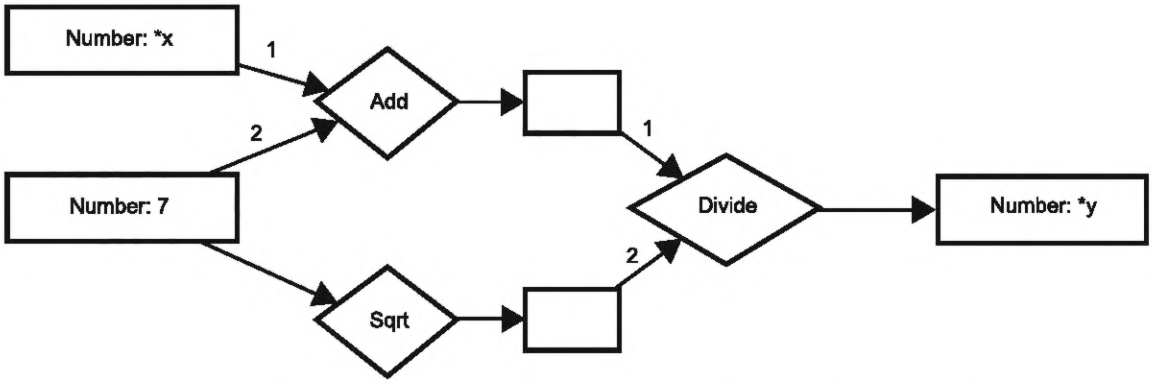

Рисунок Б.3 — Функции CL, представленные узлами действующими субъектами

В CGIF действующие субъекты представлены в виде отношений с двумя типами дуг: последовательность входных дуг и последовательность выходных дуг, разделенных вертикальной чертой:

[Number: *x] [Number: *y] [Number: 7]

(Add ?x 7 | [*u]) (Sqrt 7 | [*v]) (Divide ?u ?v | ?y)

В форме отображения входные дуги Add и Divide имеют номера 1 и 2, которые обозначают порядок записи дуг в CGIF. Ниже приведен соответствующий синтаксис CLIF:

(exists ((x Number) (y Number))

(and (Number 7) (= y (Divide (Add x 7) (Sqrt 7)))))

Для представления меток кореферентности *u и *v не требуются переменные CLIF, поскольку функциональная запись, используемая в CLIF, отображает соединения напрямую.

Все семантические особенности CL, включая возможность количественной оценки отношений и функций, поддерживаются CGIF. Например, кто-то может сказать: «Bob and Sue are related» («Боб и Сью — родственники»), но не уточнять, кем именно они друг другу приходятся. Согласно следующим высказываниям на CGIF и CLIF, существует некоторая семейная связь *r* между Бобом и Сью:

[Relation: *r] (Familial ?r) (#?r Bob Sue)
(exists ((r Relation)) (and (Familial r) (r Bob Sue)))

Концепция [Relation: *r] утверждает, что существует связь *r*. Следующие два отношения показывают, что *r* является семейной связью и что *r* существует между Бобом и Сью. В CGIF префикс # указывает на связанную метку кореферентности, используемую в качестве метки типа.

Б.1.3 Правила синтаксиса EBNF для CGIF

В соответствии с [2] для описания синтаксиса CGIF используется обозначение EBNF. Спецификации в этом приложении используют только следующее подмножество функций, определенных в [2]. В данном пункте приведен только в справочных целях, поскольку в качестве нормативного документа следует рассматривать стандарт [2].

Терминальный символ. Любая строка, заключенная в одинарные или двойные кавычки.

Пример — "This is a quoted string." 'and so is this'

Нетерминальный символ. Имя категории в синтаксическом правиле. Например, следующее синтаксическое правило содержит два нетерминальных символа: один терминальный символ «;», определяющий символ «=», символ объединения «,» и терминальный символ «;».

syntaxRule = expression, «;»;

Возможный вариант. Выражение, заключенное в квадратные скобки. От него зависит отсутствие или наличие любой строки, указанной в таком выражении.

Пример — ["This string may or may not occur."]

Итерация. Выражение, заключенное в фигурные скобки. От него зависит отсутствие или наличие нескольких экземпляров любой строки, указанной в таком выражении.

Пример — { "This string may occur many times." }

Объединение. Два или более терминов, разделенных запятыми.

"Two kinds of quotes: ", '"', " and ", '"', "."

Исключение. Два термина, разделенных знаком «-», который указывает на любую строку, обозначенную первым термином, но не вторым. В следующем примере приведена последовательность из нуля или более цифр, не содержащая «6»:

{digit} - 6

Группа. Выражение, заключенное в круглые скобки и рассматриваемое как один термин. В следующую группу входит исключение, которое определяет последовательность из одной или нескольких цифр, исключая пустой термин:

({digit} -)

Альтернативные варианты. Два или несколько объединений, разделенных вертикальными полосами.

Пример — "cat", "dog" | "cow", "horse", "sheep" | wildAnimal

Особая последовательность. Любая строка, в начале и конце которой стоят вопросительные знаки. Эти последовательности не влияют на синтаксис, указанный в правилах, но могут использоваться для копирования строк, проанализированных на основании правила, для последующего использования в правилах перезаписи, указанных в Б.1.3.

Пример — ?sqn?

Синтаксическое правило. Нетерминальный символ, за которым следует «=», и выражение, в конце которого стоит символ «;». Следующие синтаксические правила определяют синтаксис правил, используемых в приложении Б.

```
syntaxRule    = expression, ";";
expression    = alternative, {"|" alternative} | term, "-", term;
alternative   = term [variable], {"," term [variable]};
term  = terminal | nonterminal | "[", expression, "]"
| "{", expression, "}" | "(", expression, ")" | empty;
terminal    = '"', ({character - '"'} - empty), '"'
| "'", ({character - '"'} - empty), "'";
nonterminal    = identifer;
variable     = «?»,identifer, «?»;
identifier    = letter, {letter | digit | «_»};
empty       = ;
```

Эти правила определяют подмножество синтаксических правил, определенных в [2] (8.1). В соответствии с правилами символ «,» имеет более высокий приоритет, чем «|», который, в свою очередь, имеет более высокий приоритет, чем «=». Круглые скобки можно использовать для обнуления приоритета или объединения в более очевидные группы.

Б.1.4 Обозначение для правил перезаписи

Б.1.4.1 Общие положения

Синтаксис как основного (см. Б.2), так и расширенного CGIF (см. Б.3) определяется в расширенной форме Бэкуса — Наура (EBNF), как указано в [2]. Для уточнения перевода из основного CGIF в CL в Б.2 используется комбинация правил EBNF и математических обозначений, дополненная английским языком. Для уточнения перевода преобразование из расширенного CGIF в основной CGIF в Б.3 используется комбинация правил EBNF из Б.1.4 и правил перезаписи из Б.1.4.2. Правила синтаксиса в приложении Б предполагают этап лексического анализа, на котором текст разделяется на токены в соответствии ГОСТ 33707—2016 (по лексическим токенам). Следовательно, в любой точке, где в правиле EBNF встречается запятая, во входном тексте может встретиться ноль, один или несколько символов пробела.

Б.1.4.2 Правила трансформации

Каждое правило преобразования должно определять функцию, которая анализирует входную строку и возвращает последовательность из одной или нескольких выходных строк. Правило трансформации должно состоять из трех частей: заголовка, синтаксического правила, определенного в Б.1.3, и нуля, одного или нескольких правил перезаписи. Первая строка в заголовке должна содержать имя функции, которое также должно быть именем нетерминального символа, определенного синтаксическим правилом. Заголовок также должен содержать переменную со значением входной строки, которая будет проанализирована с помощью правила, и определять последовательность из одной или нескольких выходных переменных. Если входная строка была успешно проанализирована с помощью правила от начала и до конца, применяются правила перезаписи, если таковые имеются. Ниже приведены синтаксические правила, определяющие синтаксис правил трансформации; в качестве начального символа выступает transRule.

```
transRule      =  header, syntaxRule, {rewriteRule}, «end», «;»;
header         =  nonterminal, «(«, variable, «)», «->»,
                  variable, {«,» variable};
rewriteRule    =  assignment | conditional;
assignment     =  variable, «=», rewriteExpr, «;»;
conditional    =  "if", condition, ({rewrite rule} - empty),
                  {"elif", condition, ({rewrite rule} - empty)},
                  ["else", ({rewrite rule} - empty)], "end;"
condition      =  "(", test, {"&", test}, ")";
test           =  rewriteTerm, ["~"], "=", rewriteTerm;
испытание      =  rewriteTerm, ["~"], "=", rewriteTerm;
rewriteExpr    =  rewriteTerm {"," rewriteTerm};
rewriteTerm    =  terminal | variable | funTerm;
funTerm        =  identifer, "(", [funTerm, {"," funTerm}, ")";
```

Следующие нетерминальные символы из [2] должны быть определены аналогично Б.1.3: syntaxRule, terminal, nonterminal, variable, identifier, empty.

Функция, которую определяет правило трансформации, должна преобразовывать входную строку в последовательность значений выходных переменных путем копирования подстрок из входных данных и применения правил перезаписи для трансформации таких строк. При этом подлежит применению описанная ниже процедура.

В соответствии с правилами синтаксиса для анализа входной строки может быть использован любой алгоритм синтаксического анализа. В начале анализа всем переменным, которые встречаются в правиле трансформации, следует присвоить значение пустой строки. Хотя некоторые алгоритмы могут присваивать значения переменным на этапе анализа, семантика не должна содержать требований об обеспечении доступности этих значений для применения любых правил перезаписи до тех пор, пока синтаксический анализ не будет завершен полностью.

Любая переменная *x* в синтаксическом правиле должна стоять сразу после определенного термина *t* в том же правиле. Между *t* и *x* не должно стоять запятых и других разделяющих символов. Значение, присвоенное *x*, должно представлять собой подстроку *s* входной строки, сопоставленной с шаблоном, указанным *t*. Если альтернативный вариант, в котором встречается *t*, не был принят, или если *t* соответствует пустой строке, значение *x* должно быть пустым.

Если в параметрах импликации не пропущены одно или несколько правил перезаписи, то после завершения синтаксического анализа последовательно выполняются правила перезаписи, следующие за синтаксическим правилом.

В ходе соотнесения значения терминалов, переменных и функциональных терминов в правой части правила должны быть объединены в том порядке, в котором они записаны. Полученная в результате строка должна быть присвоена как значение переменной в левой части правила.

Условие, которое встречается в ходе импликации, представляет собой один или комбинацию нескольких тестов на равенство или неравенство значений двух терминов. Пустой термин, записанный как пробел, имеет в качестве значения пустую строку. Следовательно, условие (?x?= & ?y?~=) должно быть истинным, только если ?x? имеет пустое значение, а ?y? — не пустое.

При импликации условия для параметров if, elif и else должны оцениваться последовательно (условие else всегда должно выполняться.) При обнаружении первого истинного условия правила перезаписи, следующие за этим условием, будут последовательно выполняться до следующего значения elif, else или end для этого правила. После этого выполняется правило перезаписи, которое располагается после маркера end для этого условия (если таковое есть).

По достижении маркера end для правила трансформации выполнение правил следует прекратить. Тогда значение функции, указанной в заголовке, должно представлять собой последовательность значений всех выходных переменных. Любая выходная переменная, которой не было присвоено значение, должна иметь значение пустой строки. Любая выходная переменная, имеющая тот же идентификатор, что и определенная переменная в синтаксическом правиле, должна иметь значение, присвоенное ей во входной строке. В ходе соотнесения значения переменных не должны изменяться после присвоения.

Согласно этой спецификации некоторые правила трансформации могут не включать правила перезаписи. Следующее правило, например, определяет функцию идентификации, результат которой идентичен входным данным:

```
identity(?s?) -> ?t?;
identity = {character} ?t?;
end;
```

Входная строка *s* анализируется на основании синтаксического правила как строка с нулем или большим количеством символов. Эта строка присваивается *t*, т. е. результату применения функции.

Значение, присвоенное переменной в результате синтаксического анализа, всегда является одной из подстрок входных данных. За исключением функции идентичности, выходные значения, сгенерированные правилами перезаписи для любой синтаксической категории, часто сильно отличаются от подстрок входных данных. Например, правило трансформации «egation» переводит отрицание из расширенного формата CGIF в основной:

```
negation(?b?) -> ?ng?;
negation = "~[", [comment] ?cm?, CG ?x?, [endComment] ?ecm?, "]";
?ng? = "~[", ?cm?, CG(?x?), ?ecm?, "]";
end;
```

Строки для начального комментария *cm* и конечного комментария *ecm* без изменений подлежат копированию из входных данных в выходные. Но вложенный CG, чья входная строка *x* имеет расширенный формат CGIF, сильно отличается от выходных данных основного формата CGIF CG (*x*). Правила трансформации для синтаксических категорий расширенного формата CGIF ведут себя как компиляторы, которые переводят входные строки из категорий расширенного формата CGIF в выходные строки основного формата CGIF.

Б.1.4.3 Функции, используемые в правилах перезаписи

Любая функция, определенная правилом трансформации, может использоваться в правиле перезаписи. Ее можно даже рекурсивно использовать в том же правиле трансформации, которое ее содержит. В дополнение к функциям, определенным правилами трансформации, следующие семь функций должны быть доступны при обработке строк или последовательностей в любом правиле перезаписи:

- first(*s*) должна возвращать первый или единственный элемент последовательности *s*. Если length(*s*)="0", first(*s*) должна быть пустой;

- gensym() должна возвращать представляющую CGname строку, которая должна отличаться от остальных CGname в текущем тексте. При каждом вызове gensym() возвращаемая строка также должна отличаться от всех возвращенных ранее строк;

- length(*s*) должна возвращать длину последовательности *s* в виде строки из одного или нескольких символов, представляющих десятичные цифры длины. Если *s* пустая, length(*s*) должна равняться «0». Если *s* — одиночный элемент, length(*s*) должна равняться «1»;

- map(*f,s*) должна применять функцию *f* к каждому элементу последовательности *s*, чтобы отобразить последовательность значений *f*(*x*) для каждого *x* в *s*;

- second(*s*) должна возвращать второй элемент последовательности *s*. Если length(*s*)<"2", second(*s*) должна быть пустой;

- substitute(*s,t,x*) должна возвращать результат замены строки *s* для каждого экземпляра строки *t* в строке *x*. Если *t* отсутствует в *x*, substitute(*s,t,x*) должна равняться *x*;

- third(*s*) должна возвращать третий элемент последовательности *s*. Если length(*s*)<"3", third(*s*) должна быть пустой. Английская фраза «CG name» должна относиться ко всем синтаксическим токенам категории «CGname».

Б.2 Основной синтаксис и семантика CG

Б.2.1 Действующий субъект

Определение: концептуальное отношение *ac*=(*r*,*s*), в котором *r* должна выступать в роли ссылки под названием «метка» типа *ac* и последовательности дуги *s*=s_1,s_2 и должна состоять из последовательности дуг s_1, *входных дуг*, и отдельной дуги s_2, выходной дуги.

CL: *cg2cl*(*ac*) должно быть уравнением *eq*: первый термин *eq* — это имя *cg2cl*(s_2), второй терм *eq* — функциональный термин с оператором *cg2cl*(*r*) и последовательностью терминов *cg2cl*(s_1) с необязательным маркером последовательности *sqn*.

CGIF:

actor = "(", [comment], ["#", "?"], CGname, arcSequence, "|", arc,

[endComment], ")";

Как и другие концептуальные отношения, узел действующего субъекта заключен в круглые скобки. Символ # должен обозначать связанную метку кореферентности, которая используется в качестве метки типа.

Комментарий: хотя действующий субъект определяется как частный случай концептуального отношения, основной синтаксис CG позволяет связать его только с одной выходной дугой, чтобы субъекта можно было сопоставить с функцией CL. В конце входных дуг могут находиться маркеры последовательности, но в выходных дугах использовать такие маркеры запрещено. Синтаксис расширенного формата CGIF позволяет участникам иметь любое количество выходных дуг.

Б.2.2 Дуга

Определение: ссылка *ar*, которая встречается в последовательности дуг определенного концептуального отношения.

CL: *cg2cl*(*ar*) должна быть именем *n* без маркера ссылки *ar*.

CGIF:

arc = [comment], reference;

Комментарий: функция *cg2cl* сопоставляет дугу с именем ссылки и пропускает все маркеры, которые выделяют связанную метку.

Б.2.3 arcSequence

Определение: пара *as*=(*s*,*sqn*), состоящая из последовательности *s* нуля или большего количества дуг, за которой следует необязательный маркер последовательности *sqn*.

CL: *cg2cl*(*as*) должна быть последовательностью терминов *ts*=*cg2cl*(*s*) и маркером последовательности *sqn*, если он присутствует в *as*. Последовательность терминов *ts* должен иметь вид map(*cg2cl*,*s*), где *map* — это функция, которая применяет *cg2cl* к каждой дуге последовательности, чтобы извлечь имя, которое становится соответствующим элементом последовательности *ts*.

CGIF:

arcSequence = {arc}, [[comment], "?", seqmark];

Любой маркер последовательности в последовательности дуги *as* должен быть идентичен маркеру последовательности в определенной концепции существования, которая непосредственно содержится в контексте с действующим субъектом или концептуальным отношением, в качестве которого указана последовательность дуги *as*.

Комментарий: Возможность наличия маркера последовательности в последовательности дуг подразумевает, что концептуальное отношение может иметь разное количество дуг.

Б.2.4 Комментарий

Определение: строка *cm*, которая не должна влиять на семантику любого выражения *x* CGIF, в котором присутствует *s*.

CL: *cg2cl*(*cm*) должна быть подстрокой *s* в *cm* без разделителей «/*» и «*/» комментария или начального символа «;» в конечном комментарии. Строка *s* должна быть включена в представление CL для комментария, а также должна быть связана с синтаксическим выражением CL, в которое переводится выражение *x* CGIF. Синтаксические правила для комментария и конечного комментария в основном и расширенном форматах CGIF идентичны.

CGIF:

comment = "/*", {(character-"*") | ["*", (character-"/")]}, ["*"], "*/";

endComment = ";", {character - ("]" 1 ")")};

Строка с разделителями «/*» и «*/», не должна содержать подстроки «*/». Строка конечного комментария может содержать любое количество символов «;», но не может содержать «]» и «)».

Комментарий: комментарий может находиться непосредственно после открывающей скобки любого концепта, действующего субъекта или концептуального отношения, непосредственно перед любой дугой или в произвольном порядке вместе с концепциями и концептуальными отношениями любого концептуального графа. Конечный комментарий может находиться непосредственно перед закрывающей скобкой любого концепта или перед закрывающей круглой скобкой любого концептуального отношения или действующего субъекта. Поскольку синтаксис комментариев в основном и расширенном форматах CGIF идентичен, дополнительные синтаксические правила для комментариев в Б.3 не требуются.

Б.2.5 Концепт

Определение: пара *c*=(*R*,*g*), где *R* должна быть определяющей меткой или набором из нуля или большего количества ссылок, а *g* — концептуальным графом, который, как ожидается, содержится непосредственно в *c*.

CL: *cg2cl*(*c*) должна быть высказыванием *s*, определяемым одним из первых трех представленных ниже вариантов.

Контекст. Если *R* имеет пустое значение, то *s*=*cg2cl*(*g*). В этом случае *c* следует называть *контекстом*.

Существование. Если *g* имеет пустое значение, а *R* — это определяющая метка, то высказывание *s* должно быть количественным высказыванием типа существования с набором имен {*cg2cl*(*R*)} и основным текстом, состоящим из логического высказывания типа конъюнкция и нулевых элементов. В этом случае *c* следует называть концептом существования.

Кореферентность. Если *g* имеет пустое значение, а *R* — это набор из одной или нескольких ссылок, тогда *r* может быть любой ссылкой в *R*. Высказывание *s* должно быть логическим высказыванием типа конъюнкция, компоненты которого представляют собой набор уравнений с первым членом *cg2cl*(*r*) и вторым членом *cg2cl*(*t*) для каждой ссылки *t* в *R*-{*r*}. В этом случае *c* следует называть концептом кореферентности.

Синтаксически неверно. Случаи, когда *g* и *R* не имеют пустых значений, не допускаются в основном синтаксисе CGIF, поэтому процедура перевода в CL не определена.

CGIF:

```
concept = context | existentialConcept | coreferenceConcept;
context = "[", [comment], CG, [endComment], "]";
existentialConcept = "[",      [comment], "*", (CGname | seqmark),
                               [endComment], "]";
coreferenceConcept =           [comment], ":", {reference}-,
"[",                           [endComment], "]";
```

Контекст должен быть концептом, содержащим CG; если CG имеет пустое значение, контекст считается пустым, даже если содержит один или несколько комментариев. Любой комментарий, который находится сразу после открывающей скобки, должен быть частью концепта; все остальные комментарии должны быть частью вложенного CG. Концепт кореферентности должен содержать одну или несколько констант или связанных меток кореферентности; в EBNF итерация, за которой следует знак минус, после которого ничего не стоит, указывает по крайней мере одну итерацию.

Комментарий: контекст представлен парой скобок, которые служат для ограничения области кванторов вложенного CG; пустой контекст [ ] переводится в CLIF как (and) и по определению является истинным. Концепт существования представлен концептом [*x], который переводится в CLIF как (exists (x) (and)); это высказывание утверждает, что существует определенный *x*. Концепт кореферентности представлен концептом, которая содержит набор констант или связанных меток кореферентности, например [: ?x Cicero Tully ?abcd], который переводится в CLIF как конъюнкция уравнений:

(and (= x Cicero) (= x Tully) (= x abcd))

Концепт кореферентности с одной ссылкой, например [:?x], стал бы пустым союзом (and). Поскольку этот концепт не имеет семантического воздействия, его можно удалить.

Б.2.6 Концептуальный граф (CG)

Определение: тройная переменная g=(C,R,A), где C — это набор концептов, R — набор концептуальных отношений, а A — набор дуг, должен состоять только из всех дуг, которые встречаются в последовательности дуг определенного концептуального отношения в R. Если C и R имеют пустые значения, A также остается пустым, а g называется пустым концептуальным графом.

CL: Пусть *E* будет подмножеством *C* концепции существования, а *X* — множеством всех концептов, концептуальных отношений и отрицаний *g*, кроме представленных в *E*.

Пусть *B* — это логическое высказывание типа конъюнкции с компонентами, состоящими из всех высказываний *cg2cl*(*x*) для каждого *x* в *X*.

Если *E* имеет пустое значение, то *cg2cl*(*g*) представляет собой *B*.

Если *E* имеет не пустое значение, то *cg2cl*(*g*) является количественным высказыванием типа существования с набором имен, состоящим из CGname определяющей метки кореферентности каждого *e* в *E* с основным текстом *B*.

CGIF:

CG = {concept | conceptualRelation | negation | comment};

Концептуальный граф состоит из неупорядоченного набора концептов, концептуальных отношений, отрицаний и комментариев. Формально отрицание — это пара из концепта и концептуального отношения, которые в CGIF никогда не разделяют.

Комментарий: согласно настоящему стандарту каждый CG сопоставляется либо с количественным высказыванием типа существования, либо с логическим высказыванием типа конъюнкции. Если конъюкция имеет только один компонент, то высказывание может быть упрощено до равенства, атомарного или логического высказывания типа отрицания. Если *g* имеет пустое значение, оно переводится в CLIF как (and) и по определению является истинным. Хотя отсутствует требование упорядочивать узлы CG, определенное программное обеспечение, обрабатыва-

ющее CGIF, может работать более эффективно, если определяющие метки кореферентности будут располагаться перед соответствующими связанными метками. Самый простой способ соблюсти это условие — переместить концепты существования на передний план любого контекста.

Б.2.7 Концептуальное отношение

Определение: Пара *cr*=(*r*,*s*), в которой *r* должна выступать в роли ссылки под названием *метка типа cr*, а *s* — в роли последовательности дуг.

CL: *cg2cl*(*ac*) должна быть атомарным высказыванием с предикатом *cg2cl*(*r*) и последовательностью терминов *cg2cl*(*s*).

CGIF:

```
conceptualRelation = ordinaryRelation | actor;
                                [comment], [ "#", "?"], CGname,
ordinaryRelation =
                        arcSequence,
 "(",
                                [endComment], ")";
```

Обычное концептуальное отношение содержит только одну последовательность дуг. Действующий субъект разбивает последовательность дуг на две подпоследовательности. Связанная метка кореферентности, которая используется в качестве метки типа, должна начинаться со строки «#?» или «#?».

Комментарий: за счет того, что метка типа концептуального отношения может быть связанной меткой, CGIF сохраняет возможности CL для выполнения количественной оценки отношений и функций. В качестве примера см. CGIF в завершающей части Б.1.2, где описано высказывание «Bob and Sue are related».

Б.2.8 Отрицание

Определение: пара *ng*=(*c,cr*), в которой *c* должно быть концептом, а *cr* — концептуальным отношением, метка типа *r* которого должна быть константой с CGname Neg. Пара (*c, cr*) должна рассматриваться как единое целое.

CL: *cg2cl*(*ng*) должна быть логическим высказыванием типа отрицания с компонентом *cg2cl*(*g*).

CGIF:

```
negation = "~", context;
```

Отрицание должно начинаться с символа «~». Хотя отрицание формально определяется как пара, состоящая из контекста и концептуального отношения, элементы пары не должны выражаться в CGIF в качестве отдельных узлов.

Комментарий: отрицание отрицает предложение, сформулированное вложенным концептуальным графом *g*. В качестве примера см. CGIF на рисунке Б.2. Отрицание пустого CG, записанное в форме ~[ ], всегда ложно; соответствие в формате CLIF: (not (and)).

Б.2.9 Ссылка

Определение: пара *r*=(*m*,*n*), где *n* является именем CG, а *m* — *маркером*, который обозначает *константу* или *связанную метку*.

CL: *cg2cl*(*r*) должна быть именем *n*. Маркер *m* должен иметь значение «?» для связанной метки и пустой строки «» для константы.

CGIF:

```
reference = ["?"], CGname;
```

Этот синтаксис ссылок идентичен основному и расширенному форматам CGIF. Любое имя CG, которое состоит из namesequence в кавычках, трансформируется в имя CL путем удаления кавычек. Все остальные имена CG идентичны соответствующим именам CL. Маркеры последовательности в CLIF и CGIF идентичны.

Комментарий: Поскольку ссылки в основном и расширенном форматах CGIF идентичны, в Б.3 не приведены дополнительные синтаксические правила для ссылок.

Б.2.10 Область применения

Определение: набор контекстов *S*, связанных с концептом *x*, который имеет определяющую метку с именем CG *n*.

Следующие термины используют при определении ограничений для определяющих меток в основном и расширенном форматах CGIF:

- константа, имя CG без префикса;
- связанная метка корефенетности, имя CG с префиксом «?»;
- связанная метка последовательности, маркер последовательности с префиксом «?»;
- связанная метка, связанная метка корефенетности или связанная метка последовательности;
- определяющая метка корефенетности, имя CG с префиксом «*»;
- определяющая метка последовательности, маркер последовательности с префиксом «*»;
- определяющая метка, определяющая метка корефенетности или определяющая метка последовательности.

Согласно этому определению, определяющая метка последовательности должна начинаться со строки «*…», а связанная метка последовательности — со строки «?...».

Ограничения: глагол «содержит» следует определять как транзитивное замыкание отношения «непосредственно содержит». Он должен удовлетворять следующим ограничениям в основном и расширенном форматах CGIF.

Если контекст *c* непосредственно содержит концептуальный граф *g*, то *c* непосредственно содержит каждый узел *g* и каждый компонент этих узлов, за исключением тех, которые содержатся в определенном контексте *g*.

Если контекст *c* непосредственно содержит контекст *d*, то *c* опосредованно содержит все, что содержит *d*.

Фраза «*c* содержит *x*» является синонимом «*c* непосредственно или опосредованно содержит *x*».

Если концепт *x* с определяющей меткой с именем *n* непосредственно содержится в определенном контексте *c*, то *c* не должен содержать других контекстов, кроме *x* с определяющей меткой с тем же именем CG *n*. Кроме того, *c* должен находиться в области применения S, связанной с концептом *x*.

Если контекст *c* находится в области применения S, связанной с концептом *x*, то любой контекст *d,* непосредственно содержащийся в *c*, также должен содержаться в области применения S, если *d* непосредственно не содержит концепт *y* с определяющей меткой с тем же именем CG, что и у определяющей метки *x*.

Все связанные метки с именем CG *n* должны находиться в области, связанной с неким концептом с определяющей меткой с именем CG *n*.

Константы с именем CG *n* не должны находиться в области, связанной с неким концептом с определяющей меткой с именем CG *n*.

Эти ограничения гарантируют, что в каждом высказывании s CGIF перевод *cg2cl*(s) будет соответствовать ограничениям CL на область применения кванторов. Поскольку ограничения на область применения идентичны в базовом и расширенном форматах CGIF, дополнительные ограничения в Б.3 отсутствуют.

Б.2.11 Текст

Определение: контекст *c*, который непосредственно или опосредованно не содержится ни в каком контексте.

CL: *cg2cl*(*c*) должна быть текстом, состоящим из предложения *cg2cl*(*g*), где *g* — концептуальный граф, непосредственно содержащийся в *c*. Если имя CG *n* встречается непосредственно перед *g* в варианте CGIF контекста *c*, то *n* должно быть именем текста CL.

CGIF:

text = “[“, [comment], “Proposition”, “:”, [CGname], CG,

[endComment], "]";

Поскольку текст не содержится ни в каком контексте, его также следует называть внешним контекстом.

Комментарий: это синтаксическое правило использует синтаксис расширенного формата CGIF, который позволяет контексту иметь метку типа и имя CG. Поскольку синтаксис основного формата CGIF является подмножеством расширенного, текст в основном формате CGIF может быть использован любым процессором, который поддерживает расширенный формат CGIF. Контекстные скобки могут использоваться для группировки понятий и отношений текста в блоки, которые соответствуют предложениям CLIF. Данная группировка используется для удобства и не влияет на семантику.

Б.3 Синтаксис расширенного формата CGIF

Б.3.1 Общие положения

Расширенный формат CGIF — это надмножество основного формата CGIF, и каждое синтаксически правильно построенное высказывание в основном формате CGIF также является синтаксически правильным и в расширенном. Его наиболее характерной особенностью является возможность выбора метки типа или выражения типа в левой части любого концепта. Помимо типов, расширенный формат CGIF отличается следующими дополнительными особенностями:

- больше вариантов концептов, включая кванторы всеобщности;
- логические контексты для представления операторов or, if и iff;
- возможность размещения узлов концепта в последовательности дуг концептуальных отношений;
- возможность импорта текста в текст.

Эти особенности позволяют сделать предложения более краткими, удобочитаемыми и подходящими в качестве исходного языка для переводов с естественных языков и других диалектов CL, включая CLIF. Однако ни один из них не расширяет выразительную силу CGIF за пределы основного CG, поскольку семантика каждой расширенной функции определяется ее переводом в основной формат CGIF, семантика которого зависит от перевода в CL.

В Б.3 дано определение реального синтаксиса расширенного формата CGIF и приведены переводы всех расширенных функций в основной формат CGIF. Этот перевод указывает на функцию *CG*, которая переводит любое высказывание в расширенном формате CGIF в семантически эквивалентное высказывание CG(*s*) в основном формате CGIF. Комбинированные функции *g2cl*(CG(*s*)) переводят *s* в логически эквивалентное высказывание с абстрактным синтаксисом CL.

Функцию CG и функции других категорий CGIF определяют правилами трансформации, обозначение которых указано в Б.1.4.1. Две категории, комментарии и ссылки, имеют идентичный синтаксис в основном и расширенном форматах CGIF; для любого комментария *cm* в расширенном формате CGIF — comment(*cm*)=*cm*; для любой ссылки *r* — reference(*r*)=*r*. В любой другой категории X основного формата CGIF строки категории X являются соответствующим подмножеством строк расширенного формата CGIF той же категории.

Поскольку определения в Б.2 описывают абстрактный синтаксис концептуального графа и его сопоставление с абстрактным синтаксисом CL, использовались независимые от обозначения конструкции, такие как множества. Приведенные ниже определения описывают сопоставление реального синтаксиса расширенного формата

CGIF с реальным синтаксисом основного формата CGIF. Следовательно, они определены в терминах строк и трансформирующих их функций.

Б.3.2 Действующий субъект

Определение: строка *ac*, которая должна содержать комментарий *cm*, ссылку *r*, называемую меткой типа, последовательность дуг s_1 — входных дуг, последовательность дуг s_2 — *выходных дуг* и необязательный конечный комментарий *ecm*. Выходные дуги s_2 не должны содержать маркера последовательности.

Перевод: концептуальный граф *g*.

```
actor(?ac?) -> ?g?;
actor        = "(", [comment] ?cm?, (["#", "?"], CGname) ?r?,
arcSequence ?s1?, "|", {arc} ?s2?, [endComment] ?ecm?, ")";
?z1?         = frst(arcSequence(?s1?));
?z2?         = frst(arcSequence(?s2?));
?sqn?        = third(arcsequence(?s1?));
if (length(?s2?)="0")
?cr? = "(", ?cm?, ?r?, ?z1?, ?sqn?, ?ecm?, ";0-output actor", ")";
elif (length(?s2?)="1")
?cr? = "(", ?cm?, ?r?, ?z1?, ?sqn?, "|", ?z2?, ?ecm?, ")";
else ?cr? = "(", ?cm?, ?r?, ?z1?, ?sqn?, "/*|*/", ?z2?, ?ecm?, ")";
end;
?g? = second(arcSequence(?s1?)), second(arcSequence(?s2?)), ?cr?;
end;
```

Если *s2* не имеет выходных дуг, *cr* должно быть обычным концептуальным отношением, как указано в Б.3.7, но чтобы показать, что *cr* является производным от действующего субъекта, вставляется конечный комментарий «0-output actor». Если *s2* имеет одну выходную дугу, *cr* должно быть действующим субъекта, но *cr* отличается от *ac*, потому что дуги переводятся в основной формат CGIF. Если *s2* имеет две или более выходных дуг, *cr* должно быть обычным концептуальным отношением, но при этом для разделения входных и выходных дуг добавляется комментарий «/*|*/». Последнее правило перезаписи помещает *cr* после всех концептуальных графов, полученных из последовательностей дуг.

Комментарий: например, при использовании правил трансформации с действующими субъектами, дугами, последовательностями дуг и концептами следующий узел действующего субъекта:

(IntegerDivide [Integer: *x] [Integer: 7] | *u *v)

будет переведен в концептуальный граф с шестью узлами, состоящий из трех концептов и трех концептуальных отношений:

[*x] (Integer ?x) (Integer 7) [*u] [*v]

(IntegerDivide ?x 7 /*|*/ ?u ?v)

Комментарий «/*|*/» не имеет семантического действия в основном формате CGIF и CL, но, если его сохранить, этот комментарий позволит выполнить обратное сопоставление с расширенным форматом CGIF для разграничения входных и выходных дуг. Если такое разграничение требуется в определенном приложении, можно использовать аксиомы, чтобы установить функциональные зависимости выходных дуг от входных. Например, отношение CL, которое возникает в результате перевода действующего субъекта типа IntegerDivide, удовлетворяет следующему ограничению, указанному в CLIF:

(exists (Quotient Remainder) (forall (x1 x2 x3 x4)

(iff (IntegerDivide x1 x2 x3 x4)

(and (= x3 (Quotient x1 x2)) (= x4 (Remainder x1 x2))))))

Это высказывание утверждает, что существуют функции Quotient и Remainder, которые определяют значения третьего и четвертого аргументов отношения IntegerDivide. Правила перевода не генерируют эту аксиому автоматически, но ее можно выразить высказыванием CGIF, которое будет переведено в высказывание CLIF:

[*Quotient] [*Remainder]

[[@every*x1] [@every*x2] [@every*x3] [@every*x4]

[Equiv: [Iff: (IntegerDivide ?x1 ?x2 | ?x3 ?x4)]

[Iff: (#?Quotient ?x1 ?x2 | ?x3) (#?Remainder ?x1 ?x2 | ?x4)]]]

Чтобы показать, что кванторы существования для [*Quotient] и [*Remainder] имеют приоритет над квантификаторами всеобщности для четырех аргументов, узлы концептов заключаются в контекстные скобки с кванторами всеобщности.

Б.3.3 Дуга

Определение: строка *ar*, которая должна содержать необязательный комментарий *cm* и ссылку *r*, определяющую метку с именем CG *n* или концепт *c*.

Перевод: Пара (*x*,*g*), состоящая из дуги *x* и концептуального графа *g*.

arc(?ar?) -> ?x?, ?g?;
arc = [comment] ?cm?, (reference ?r? | "*", CGname ?n? | concept ?c?);
if (?r?~=) ?x? = ?ar; ?g? = ;
elif (?n?~=) ?x? = ?cm?, "?", ?n?; ?g? = "[*", ?n?, "]";
else ?x? = ?cm?, frst(concept(?c?));
?g? = third(concept(?c?));
end; end;

Если *ar* является ссылкой, *x* должен соответствовать значению *ar*, а *g* — иметь пустое значение. Если *ar* содержит определяющую метку, *x* должен быть результатом замены маркера * в *ar* на ?, а *g* — концептом [**n*]. Если *ar* содержит концепт *c*, *x* должен быть результатом замены концепта *c* в *ar* на ссылку *r*, а *g* — third(concept(*c*)).

Комментарий: Например, если дуга *ar* — это [Integer], значением concept([Integer]) будет имя CG, например g00023, а дуга ([Integer]) будет парой, состоящей из ссылки ?g00023 и концептуального графа [*g00023] (Integer ?g00023).

Б.3.4 arcSequence

Описание: строка *as,* которая должна содержать последовательность *s* из нуля, одной или нескольких дуг, за которой следует необязательный маркер последовательности *sqn*.

Перевод: тройная переменная (*rs,g,sqn*), состоящая из последовательности ссылок *rs*, концептуального графа *g* и маркера последовательности *sqn*.

arcSequence(?as?) -> ?rs?, ?g?, ?sqn?;
arcSequence = {arc} ?s?, [[comment], "?", seqmark] ?sqn?;
?rs? = map(frst,map(arc,?s?));
?g? = map(second,map(arc,?s?));
end;

Комментарий: функция map(arc,?s?) применяет arc к каждой дуге *s*, чтобы создать последовательность пар, состоящую из ссылки и концепта. Затем map(frst,map(arc,?s?)) извлекает последовательность ссылок из первого элемента каждой пары. Наконец, map(second,map(arc,?s?)) извлекает последовательность концептов из второго элемента каждой пары. Возможность наличия маркера последовательности в последовательности дуг подразумевает, что концептуальное отношение может иметь разное количество дуг. Действующий субъект может иметь разное количество входных дуг, но количество выходных дуг должно быть фиксированным. Следовательно, выходные дуги не должны иметь маркера последовательности.

Б.3.5 Логическое значение

Определение: строка *b*, которая должна содержать контекст *bc*, который, в свою очередь, не должен непосредственно содержать ссылку или определяющую метку. Контекст *bc* должен либо иметь префикс «~» и не иметь метку типа, либо не иметь префикса и иметь одну из следующих констант в качестве метки типа: Either, Equiv, Equivalence, If, Iff, Then.

Перевод: отрицание *ng*, которое должно иметь значение negation(*b*), eitherOr(*b*), ifThen(*b*), or equiv(*b*).

boolean = negation | eitherOr | ifThen | equiv;
negation(?b?) -> ?ng?;
negation = "~[", [comment] ?cm?, CG ?x?, [endComment] ?ecm?, "]";
?ng? = "~[", ?cm?, CG(?x?), ?ecm?, "]";
end;
ifThen(?b?) -> ?ng?;
ifThen = "[", [comment] ?cm1?, "If", [":"], CG ?ante?,
"[", [comment] ?cm2?, "Then", [":"], CG ?conse?,
[endComment] ?ecm1?, «]», [endComment] ?ecm2?, «]»;
?ng? = «~[«, ?cm1?, CG(?ante?),
«~[«, ?cm2?, CG(?conse?), ?ecm1?, «]», ?ecm2?, «]»;
end;
equiv(?b?) -> ?ng?;
equiv = "[", [comment] ?cm1?, ("Equiv" | "Equivalence"), [":"],
"[", [comment] ?cm2?, "Iff", [":"], CG ?g1?,
[endComment] ?ecm2? «]»,
"[", [comment] ?cm3?, "Iff", [":"], CG ?g2?,
[endComment] ?ecm3? «]», [endComment] ?ecm1? «]»;
?ng? = «[«, ?cm1?, «~[«, ?cm2?, CG(?g1?),
«~[«, CG(?g2?), «]», ?ecm2?, «]», ?cm2?,
«~[«, ?cm3?, CG(?g2?),
«~[«, CG(?g1?), «]», ?ecm3?, «]», ?ecm1?, «]»;
end;
eitherOr(?b?) -> ?ng?;
eitherOr = "[", [comment] ?cm?, "Either", [":"],

```
{[comment], nestedOrs} ?ors?, [endComment] ?ecm?, «]»;
?ng? = "~[", ?cm?, nestedOrs(?ors?), ?ecm?, "]";
end;
nestedOrs(?ors?) -> ?g?;
nestedOrs = ( "[", [comment] ?cm?, "Or" ?frst?, [":"], CG ?ng?,
[endComment] ?ecm?, "]", nestedOrs ?more? | );
if (?frst?= ) ?g? = ;
else ?g? = "~[", ?cm?, CG(?ng?), ?ecm?, "]", nestedOrs(?more?);
end; end;
```

Правило для nestedOrs рекурсивно обрабатывает последовательность из нуля, одного или нескольких логических контекстов типа Or. Если *b* не содержит вложенных контекстов Or, eitherOr(b) должно иметь значение «~[ ]», что неверно; соответствующее высказывание CLIF (or) определяется как ложное.

Комментарий: область применения кванторов в любом логическом контексте должна определяться вложением их переводов в основной формат CGIF. Любая определяющая метка в контексте типа If должна иметь в своей области применения вложенный контекст типа Then. При этом ни один из двух любых контекстов, непосредственно содержащихся в контексте типа Either, Equivalence или Equiv, не должны иметь другого в своей области применения.

Б.3.6 Концепт

Определение: строка *c*, состоящая из четырех подстрок, любая или все из которых могут быть опущены: начальный комментарий *cm*, поле типа, поле объекта ссылки и конечный комментарий *ecm*.

Поле объекта ссылки *c* может содержать определяющую метку последовательности с маркером последовательности *sqn*. В таком случае поле типа *c* должно быть пустым, перед определяющей меткой последовательности может стоять значение «@every», а в поле объекта ссылки *c* не должно быть никаких ссылок и концептуальных графов.

Если *sqn* отсутствует, поле типа *c* должно содержать выражение типа *tx* и двоеточие «:» или необязательную ссылку *ty*, называемую меткой типа, и необязательное двоеточие «:». Если *sqn* отсутствует, поле объекта ссылки *c* должно содержать необязательную определяющую метку с именем CG *df* (перед которой может стоять «@every»), последовательность из нуля, одной или нескольких ссылок *rf* и концептуальный граф *g*, который может быть пустым. Если все варианты опущены, понятие *c* должно иметь вид строки «[ ]».

Перевод: тройная переменная (*r,q,g*), состоящая из ссылки или связанной метки последовательности *r*, квантора *q* («@every» или пустой строки) и концептуального графа *g*, который должен содержать по крайней мере один концепт.

```
concept = "[", [comment] ?cm?,
( (typeExpression ?tx?, ":"
| [["#" , "?"], CGname] ?ty?, [":"]),
[["@every"] ?q?, "*", CGname ?df?], {reference} ?rf?, CG ?x? | ["@every"] ?q?, "*", seqmark ?sqn? ), [endComment]
?ecm?, "]";
if (?sqn?~= ) ?r? = "?", ?sqn?; ?g1? = "[", ?cm?, "*", ?sqn?, ?ecm?];
elif (?df?~= ) ?r? = "?", ?df?; ?g1? = "[", ?cm?, "*", ?df?, ?ecm?];
if (?rf?~= ) ?g2? = "[", ":", ?r?, ?rf?, "]"; end;
elif (?rf?~= ) ?r? = frst(?rf?);
?g2? = "[", ?cm?, ":", ?rf?, ?ecm?, "]";
else ?df?  = gensym(); ?r? = "?", ?df?;
?g1? = "[", ?cm?, "*", ?df?, ?ecm?, "]";
end;
if (?tx?~= )  ?b? = frst(typeExpression(?tx?));
?gx? = second(typeExpression(?tx?));
?g3? = substitute(?r?,?b?,?gx?);
elif (?ty?~= ) ?g3? = "(", ?ty?, ?r?, ")"; end;
if (?x?~= )  ?g4? = "[", CG(?x?), "]";
end;
?g? = ?g1?, ?g2?, ?g3?, ?g4?;
end;
```

В поле типа разрешено использовать четыре варианта переменных: выражение типа *tx*, связанную метку кореферентности с префиксом «#», константу или пустую строку. После *tx* обязательно должно стоять двоеточие. Правила перезаписи перемещают функции из концепта *c* в четыре строки, которые объединяются в концептуальный граф *g*: *g*1 — это концепт существования с определяющей меткой из *c* или с меткой, созданной gensym(), если в *c* отсутствует определяющая метка или ссылка; *g*2 — это концепт кореферентности, если в *c* встречаются какие-либо ссылки; *g*3 — это концептуальное отношение с меткой типа *ty* либо концептуальный граф, созданный из выражения типа *tx*; *g*4 — это контекст, содержащий любой непустой CG *x*. Все комментарии *cm* и *ecm* помещаются в первый непустой концепт: *g*1 или *g*2.

Комментарий: чтобы проиллюстрировать перевод, высказывание «A pet cat Yojo is on a mat» («Домашний кот Йоджо лежит на лежанке») можно представить в расширенном формате CGIF с двумя концептуальными узлами в последовательности дуг концептуального отношения:

(On [@*x (Pet ?x) (Cat ?x): Yojo] [Mat])

Для создания эквивалентного основного формата CGIF из последовательности дуг удаляются концепты. Вместо них используются ссылки, связывающие их с концептами, которые расширены указанными выше правилами перезаписи. Ниже приводится результат в основном формате CGIF:

[: Yojo] (Pet Yojo) (Cat Yojo)

[*g00238] (Mat ?g00238) (On Yojo ?g00238)

Имя CG Yojo — это ссылка на первый концепт, а имя CG g00238, созданное с помощью gensym(), — на лежанку. В п. Б.3.9 приведены обсуждение выражения типа и его перевод. Перевод *cg2cl* переведет основной формат CGIF в абстрактный синтаксис, который будет выражен следующим CLIF:

(exists (g00238) (and (= Yojo Yojo) (Pet Yojo) (Cat Yojo)

(Mat ?g00238) (On Yojo ?g00238)))

Концепт кореферентности с одной ссылкой, например [: Yojo], не влияет на истинность или ложность высказывания. Если он не требуется в качестве контейнера для комментариев, его может удалить оптимизирующий компилятор.

Б.3.7 Концептуальный граф (CG)

Определение: строка *cg*, состоящая из неупорядоченной последовательности подстрок, представляющих концепты, концептуальные отношения, логические значения и комментарии.

Перевод: концептуальный граф *g*.

```
CG(?cg?) -> ?g?;
CG = {concept | conceptualRelation | boolean | comment};
if (frst(sortCG(?cg?)~= )
?g? = "~", "[", frst(sortCG(?cg?)),
"~", "[", second(sortCG(?cg?), "]", "]";
else ?g? = second(sortCG(?cg?));
end; end;
```

sortCG(*cg*) следует быть парой (*g1, g2*), где *g1* — концептуальный граф, полученный из всех связанных квантором общности концептов в *cg*, а *g2* — концептуальный граф, полученный из всех других концептов, концептуальных отношений и комментариев в *cg*.

```
sortCG(?cg?) -> ?g1?,?g2?;
sortCG = ( (concept ?c? | conceptualRelation ?x?
| boolean ?x? | comment ?x?), sortCG ?rem?
| );
if (?c?= ) ?cg2? = CG(?x?));
elif (second(concept(?c?)) = "@every")
?cg1? = third(concept(?c?));
else  ?cg2? = third(concept(?c?));
end;
?g1? = ?cg1?, frst(sortCG(?rem?)); ?g2? = ?cg2, second(sortCG(?rem?));
end;
```

Комментарий: если во входной строке отсутствуют концепты, содержащие кванторы общности, в результате должна создаваться одна строка в основном формате CGIF, объединяющая результаты перевода всех узлов независимо друг от друга. Но если входная строка содержит какие-либо концепты существования, выходная строка должна включать в себя два отрицания. Внешний контекст должен содержать переводы всех концептов существования, а внутренний — всех других узлов во входных данных.

Б.3.8 Концептуальное отношение

Определение: строка *cr*, представляющая собой обычное концептуальное отношение или действующий субъект.

Перевод: концептуальный граф *g*, который должен быть либо ordinaryRelation(*cr*), либо действующим субъектом (*cr*).

```
conceptualRelation = ordinaryRelation | actor;
ordinaryRelation(?cr?) -> ?g?;
ordinaryRelation = "(", [comment] ?cm?, (["#", "?"], CGname) ?r?,
arcSequence ?s?, [endComment] ?ecm?, ")";
?g? = second(arcSequence(?s?)),
"(", ?cm?, ?r?, frst(arcSequence(?s?)),
third(arcSequence(?s?)), ?ecm?, ")";
end;
```

Первая строка правила перезаписи извлекает концептуальный граф из последовательности дуг *s*. Вторая строка добавляет вводный комментарий, метку типа и последовательность дуг концептуального отношения. Третья строка добавляет маркер последовательности, если таковой имеется, конечный комментарий и закрывающую скобку концептуального отношения.

Комментарий: например, концептуальное отношение (On [Cat: Yojo] [Mat]) будет переведено с помощью правил для концептуальных отношений, дуг, последовательностей дуг и концептов с целью создания концептуального графа, выраженного в основном формате CGIF:

[: Yojo] (Cat Yojo) [*g00719] (Mat ?g00719) (On Yojo ?g00719)

Б.3.9 Текст

Определение: контекст *c*, который непосредственно или опосредованно не содержится ни в каком контексте. Перевод: контекст *cx*.

text(?c?) -> ?cx?;
text = “[“, [comment] ?cm?, “Proposition”, “:”, [CGname] ?n?,
CG ?g?, [endComment] ?ecm?, “]”;
?cx? = “[“, ?cm?, “Proposition”, “:”, ?n?, CG(?g?), ?ecm?, “]”;
end;

Комментарий: CGIF не содержит явного синтаксиса для модулей. Вместо этого все модули CL сначала следует переводить в текст в основном формате CLIF в соответствии со спецификацией из А.3. Затем результат этого перевода должен быть переведен в текст в расширенном CGIF в соответствии с функцией *cl2cg*, которая определена в п. Б.4.

Б.3.10 Выражение типа

Определение: строка *tx* , содержащая имя CG *n* и концептуальный граф *g*.

Перевод: пара (*b,g*), состоящая из связанной метки *b* и концептуального графа *g*.

typeExpression(?tx?) -> ?b?,?g?;
typeExpression = “@”, “*”, CGname ?n?, CG ?g?;
?b? = «?», ?n?;
end;

Если концепт *c* содержит выражение типа, правила перезаписи, которые определяют concept(*c*), используют функцию substitute(?r?,?b?,?g?) для замены части ссылок *r* для каждого экземпляра *b* в *g*.

Комментарий: выражение типа соответствует лямбда-выражению, в котором имя CG *n* определяет формальный параметр, а концептуальный граф *g* является основным текстом выражения. Если концепт *c* содержит выражение типа, правила трансформации, по которым обрабатывается *c*, должны заменять ссылку, полученную из *c*, для каждого экземпляра связанной метки ?*n*, которая встречается в *g*.

Б.4 Совместимость с CGIF

В этом приложении описан синтаксис трех диалектов CL: абстрактный синтаксис для концептуальных графов, реальный синтаксис для основного формата CGIF и реальный синтаксис для расширенного формата CGIF. Эти языки полностью совместимы с диалектами CL в том смысле, что из каждого высказывания CL можно получить семантически эквивалентное высказывание на каждом из них и наоборот. Семантическая эквивалентность устанавливается по определению: семантика каждого высказывания в расширенном формате CGIF определяется переводом в предложение в основной формат CGIF, семантика каждого предложения в основном формате CGIF — переводом в предложения на абстрактный синтаксис CG, а семантика каждого абстрактного высказывания CG — переводом на абстрактный синтаксис CL.

Для подтверждения полной совместимости в Б.4 определена функция *cl2cg*, которая должна переводить любое предложение s в CL в предложение *cl2cg*(*s*) в расширенном формате CGIF при условии сохранения значения истинности, которое *s* имеет в любой интерпретации CL. Большинство выражений CL несложно сопоставить с определенным выражением в расширенном формате CGIF. Однако перевод функциональных терминов из CL в CGIF состоит из нескольких шагов. Любое применение функции CL можно преобразовать в действующий субъект, представляющий функцию со ссылкой на некоторый концепт, который, в свою очередь, является значением этой функции. Для перевода последовательности терминов CL в последовательность дуг в расширенном формате CGIF, узел действующего субъекта должен быть заключен в узел концепта.

Например, пусть (F X1 X2) будет термином CLIF с оператором F, примененным к аргументам X1 и X2, где имена X1 и X2 связаны кванторами, а F — не связан. Когда этот термин переводится с помощью *cl2cg*, для генерации имени CG, например g00592, следует использовать функцию gensym(). При наличии префикса «?» это имя становится связанной меткой кореферентности, которая должна использоваться в качестве выходной дуги действующего субъекта, представляющего функцию F. Необходимый результат преобразования исходного термина CLIF с помощью *cl2cg*: (F ?X1 ?X2 | ?g00592). Определяющая метка *g00592 должна быть помещена в концепт, например [*g00592], а действующий субъект также должен находиться внутри этого концепта в виде вложенного концептуального графа: [*g00592 (F X1 X2 | ?g00592)]. Этот концепт должен представлять собой результат применения *cl2cg* к функциональному термину. Он может выглядеть как дуга в последовательности дуг какого-либо действующего субъекта или концептуального отношения.

Поскольку предикат отношения CL или оператор функции CL могут быть функциональным термином, для перевода предиката или оператора в концепт следует использовать ту же процедуру. Пусть ((F X1 X2) Y1 Y2) — это атомарное высказывание CLIF, предикат которого является тем же функциональным термином, что и в предыдущем примере. Следовательно, связанная метка «?g00592», которая представляет значение функции, должна быть меткой типа соответствующего концептуального отношения. Если и Y1, и Y2 связаны кванторами в CL, концептуальное отношение должно иметь вид (#?g00592 ?Y1 ?Y2). Чтобы сгенерировать отдельную синтаксическую единицу в качестве значения *cl2cg*, данное концептуальное отношение следует поместить внутрь концепта, представляющего собой функциональный термин, непосредственно перед «]»: [*g00592 (F X1 X2 | ?g00592) (#?g00592 ?Y1 ?Y2)]. Этот концепт должен представлять собой результат применения *cl2cg* к исходному атомарному высказыванию. Он может иметь вид узла концептуального графа, который является результатом перевода большего высказывания CL, содержащего исходное атомарное высказывание.

Для каждого выражения CL **E** в таблице Б.1 указано выражение в расширенном формате CGIF, которое определяет *cl2cg*(**E**). Чтобы ограничения CL для области применения квантора сохранились в переводах с помощью *cl2cg*, переводы выражений типа E13 и E14 заключаются в контекстные скобки «[» и «]». В некоторых случаях необходимость в этих скобках отсутствует, поэтому их можно игнорировать.

В первом столбце таблицы Б.1 приведены ссылки на строки в таблице 2. Во втором столбце используются метаязык и соглашения, на основании которых был определен абстрактный синтаксис CL. В третьем столбце этот метаязык объединяется с обозначением, используемым в правилах перезаписи в Б.1.4.2. Это дает возможность определить функцию *cg2cl*, которая переводит любое высказывание *s* из основного формата CGIF в логически эквивалентное высказывание *cg2cl*(*x*) на CL.

Таблица Б.1 — Сопоставление абстрактного синтаксиса CL с синтаксисом расширенного формата CGIF

Показатель	Если E является выражением CL в виде	Тогда *cl2cg*(E) =
EI	Цифра «n»	Цифра «n»
EI	Строка в кавычках «s»	Строка в кавычках «s»
EI	Интерпретируемое имя «n»	Если имя «n» не является идентификатором CG, оно должно быть заключено в кавычки. Если оно находится в кванторе некоторого высказывания CL, перед ним должен стоять префикс «*». Если оно связано с квантором, перед ним должен стоять префикс «?»
E2	Маркер последовательности *S*	*S*
E3	Последовательность терминов <T1 … Tn>, которая начинается с термина T1	Последовательность дуг: *cl2cg*(T1) … *cl2cg*(Tn)
E4	Последовательность терминов T1 … Tn, которая начинается с маркера последовательности T1	Последовательность дуг: *cl2cg*(T2), ..., *cl2cg*(Tn), *cl2cg*(T1)
E5	Термин (O T1 … Tn)	Концепт со сгенерированным именем «n», содержащий вложенный действующий субъект: "[", "*", ‘n’, "(", *cl2cg*(O), *cl2cg*(T1, … Tn), ")", "|", "?", ‘n’, ")", "]"
	Термин (CL: прокомментированная «строка» T)	Дуга с комментарием: "/*", ‘string’, "*/", *cg2cl*(T)
E6	Уравнение (= T1 T2)	CG, состоящий из одного, двух или трех концептов. Если T1 и T2 являются именами, используется один концепт: "[:", cl2cg(T1), cl2cg(T2), "]" Если T1 и T2 являются функциональными терминами, используется три концепта: cg2cl(T1), cg2cl(T2), "[", "?", ‘n1’, "?", ‘n2’, "]", где «n1» — это имя, созданное для T1, а «n2» — имя, созданное для T2
		Если Ti — функциональный термин (где i = 1 или i = 2), а другой термин Tj — имя, используется два концепта: cl2cg(Ti), "[", "?", ‘ni’, cl2cg(Tj), "]", где «ni» — это имя, созданное для Ti

Окончание таблицы Б.1

Показатель	Если E является выражением CL в виде	Тогда *cl2cg*(E) =
E7	Атомарное высказывание (P T1 … Tn)	CG, состоящий из концептуального отношения или концепта Если P — имя, используется концептуальное отношение: "(", cl2cg(P), cl2cg(T1 ...Tn), ")" Если P — функциональный термин, используется концепт: cl2cg(P), в который перед закрывающей скобкой «]» вставлено следующее концептуальное отношение: "(", 'n', cl2cg(T1 ... Tn), ")", где "n" — это имя, созданное для cl2cg(P)
E8	Логическое высказывание (not P)	Отрицание: "~", "[", cl2cg(P), "]"
E9	Логическое высказывание (and P1 … Pn)	Концептуальный граф: c12cg(Pi), ..., c!2cg{Vn)
E10	Логическое высказывание (or P1 … Pn)	Концептуальный граф: "[", "Either", "[", "Or", cl2cg[?I), "]", ..., "[", cl2cgtPn), "]", "]"
EII	Логическое высказывание (if P Q)	Концептуальный граф: "[", "If", cl2cg(P), "[", "Then", cl2cg(Q), "]", "]"
E12	Логическое высказывание (iff P Q)	Концептуальный граф: "[", "Equiv", ":", "[", "Iff", cl2cg(P), "]", "[", "Iff", cl2cgm"]", "]"
	Высказывание (CL: прокомментированная «строка» T)	Комментарий и концептуальный граф: "/*", 'string', "*/", cl2cg(P)
E13	Количественное высказывание (forall (N1… Nn) B), где переменные от N1 до Nn — это имена или маркеры последовательности	Концептуальный граф: "[", "[", "@every", "*", cl2cg(NT\, "]", ..., "[", "@ every", "*", cl2cg(Nn), "]", cl2cg(B), "]"
E14	Количественное высказывание (exists (N1… Nn) B), где переменные от N1 до Nn — это имена или маркеры последовательности	Концептуальный граф: "[", "[", "*", cl2cg(NV), "]", ..., "[", "*", cl2cg(Nn), "]", d2cg{B\"]"
	Заявление (cl: comment "string")	Комментарий: "/*", 'string', "*/"
E17	Заявление (cl: imports N)	Концепт: "[", "cg_Imports", cl2cg(N), "]"
E18	Модуль с именем N, список исключений N1 ... Nn и текст T	Если M — это перевод в основной формат CL, указанный в п. A.3, используется текст: "[", "Proposition", ":", cl2cg(M), "]"
E19	Заявление (cl: text T1 … Tn)	Текст: "[", "Proposition", cl2cg(T1 ... Tn), "]"
E20	(cl: text N T1 …Tn)	Текст: "[", "Proposition", ":", cl2cg(N), cg2cl(T1 ... Tn), "]"

Для перевода из расширенного формата CGIF в основной в Б.3 используется комбинация синтаксических правил EBNF, а также правила перезаписи, указанные в Б.1.4.2 для определения функции ex2cor, которая переводит все высказывания *s* из расширенного формата CGIF в логически эквивалентные высказывания CG(*s*) в основном формате CGIF.

Приложение В
(обязательное)

Расширяемый язык разметки eXtended Common Logic Markup Language (XCL)

В.1 Общие положения

XCL — это определение XML для CL. Это предполагаемый язык для передачи CL по сети. Это прямое переложение абстрактного синтаксиса и семантики CL в форму XML.

В.2 Синтаксис XCL

Поскольку лексический синтаксис XCL аналогичен XML, синтаксис XCL описывается схемой Relax NG в компактной форме (RNC), доступ к которой обычно осуществляется в электронном виде. Для полноты информации и обеспечения стандартизации ниже полностью представлена схема RNC.

Документ XML, в котором элементы XCL смешаны с элементами из сторонних пространств имен, следует обрабатывать как корпус, в котором все ближайшие к корню элементы XCL выражают текст. Не приведенные в схеме ниже элементы и атрибуты в элементах XCL следует рассматривать как синтаксические расширения. Они не должны игнорироваться синтаксическими анализаторами и обрабатываться как комментарии.

Схемы Relax NG допускают неоднозначные определения. В случае совпадения с несколькими результатами использования схемы XCL первый из них имеет приоритет при сопоставлении с абстрактным синтаксисом.

В соответствии со Спецификацией XML, «содержимое» в документации схемы элемента означает содержимое XML. Согласно спецификации XML, содержимое XML включает символьные данные и разметку, но не содержит комментарии XML и инструкции по обработке. Атрибуты элемента не являются частью его содержимого XML.

```
default namespace = "http: //purl.org/xcl/ 2.0/ "
namespace xs = " http: //www .w 3.org/2001/XMLSchema "
## Синтаксис XCL CL, версия 2.0
## Корневые элементы
## Корневой элемент документа XCL
## может иметь вид cl: cl: Document или быть любым другим элементом, соответствующим
## шаблонам clText, clStatement ил clSentence.
start |= clDocument
start |= clText
start |= clStatement
start |= clSentence
## Документы
## Элемент cl: Document содержит семантически нейтральный
## корневой элемент, который позволяет документу XML содержать
## несколько текстов CL.
clDocument = element Document { Document.type }
Documenttype = clCommentable, clText*
## Тексты
clText |= Construct
clText |= Restrict
clText |= Import
## -Текстовые конструкции
## Элемент cl: Construct представляет собой текстовую конструкцию
## (TextConstruction) абстрактного синтаксиса.
Construct = element Construct { Construct.type }
Construct. type =
clCommentable,
```

```
## аргументы
(clText | clStatement | clSentence)*
## -Ограничения области
## Элемент cl: Restrict представляет собой текст ограничения области
## (DomainRestriction) абстрактного синтаксиса.
Restrict = element Restrict { Restrict.type }
Restricttype =
clCommentable,
## локальная вселенная дискурса
clTerm,
## основной текст
clText
## -Импорты
## Элемент cl: Import представляет собой текст импорта
## (Importation) абстрактного синтаксиса.
Import = element Import { Import.type }
Importtype =
clCommentable,
## имя
Имя
## Утверждения
clStatement |= In
clStatement |= Out
clStatement |= Titling
## -Дискурсивные утверждения
## Элемент cl:In представляет собой дискурсивное
## утверждение (DiscourseStatement) абстрактного синтаксиса.
In = element In { Discourse.type }
## Элемент cl: Out представляет собой
## утверждение вне дискурса (DiscourseStatement) абстрактного синтаксиса.
Out = element Out { Discourse.type }
Discourse.type =
clCommentable,
## последовательность терминов
clTerm+
## -Заголовки текста
## Элемент cl: Titling представляет собой озаглавливающее
## утверждение (Titling) абстрактного синтаксиса.
Titling = element Titling { Titling.type }
Titling.type =
clCommentable,
## имя
Name,
## текст
```

```
clText
## Последовательности clSentence |=
Atom clSentence |= Equal
clSentence |= And
clSentence |= Or
clSentence |= Not
clSentence |= Implies
clSentence |= Biconditional
clSentence |= Forall
clSentence |= Exists
## -Атомарные формулы
## Элемент cl: Atom представляет собой атомарное
## высказывание (Atomic) абстрактного синтаксиса.
Atom = element Atom { Atom.type }
Atom.type =
clCommentable,
## оператор
clTerm,
## последовательность аргументов
clTermSequence
## -Равенства
## Элемент cl: Equal представляет собой
## высказывание уравнения (Id) абстрактного синтаксиса.
Equal = element Equal { Equal.type }
Equal.type =
clCommentable,
## аргументы
clTerm,
clTerm
## -Конъюкции
## Элемент cl: And представляет собой
## конъюкцию (Conj) абстрактного синтаксиса.
And = element And { AndOr.type }
## Элемент cl:Or представляет собой
## дизъюнкцию (Disj) абстрактного синтаксиса.
Or = element Or { AndOr.type }
AndOr.type =
clCommentable,
## высказывания с компонентом
clSentence*
## Элемент cl: Not представляет собой
## отрицание (Neg) абстрактного синтаксиса.
Not = element Not { Not.type }
Not. type =
```

```
clCommentable,
## высказывание с отрицанием
clSentence
## Элемент cl: Implies представляет собой
## импликацию (Cond) абстрактного синтаксиса.
Implies = element Implies { Implies.type }
Implies.type =
clCommentable,
## высказывание с условием (антецедентом)
clSentence,
## высказывание с выводом (консеквентом)
clSentence
## Элемент cl: Biconditional представляет собой
## двойную импликацию (Bicond) абстрактного синтаксиса.
Biconditional = element Biconditional { Biconditional.type }
Biconditional.type =
clCommentable,
## высказывания с двумя компонентами
clSentence,
clSentence
## -Количественные оценки
## Элемент cl: Forall представляет собой
## высказывание с квантором всеобщности (UQuant) абстрактного синтаксиса.
Forall = element Forall { Forall.type }
## Элемент cl: Forall представляет собой
## высказывание с квантором существования (EQuant) абстрактного синтаксиса.
Exists = element Exists { Exists.type }
Forall.type = Quantifier.type
Exists.type = Quantifier.type
Quantifier.type =
clCommentable,
## связывающее высказывание
(Name | Marker)+,
## количественное высказывание
clSentence
## Последовательность терминов
clTermSequence = (clTerm | Marker)*
## Термины
clTerm |= Apply
clTerm |= Name
## -Функциональные термины
## Элемент cl: Apply представляет собой функциональный термин
## абстрактного синтаксиса.
Apply = element Apply { Apply.type }
```

```
Apply.type =
clCommentable,
## оператор
clTerm,
## последовательность аргументов
clTermSequence
## -Имена
## Элемент cl: Name представляет собой имя (V)
## абстрактного синтаксиса.
## Фактический символ имени, т. е. сущности,
## которая относится к лексикону, создается путем сопоставления лексического
## значения элемента cl: Name с пространством значений
## типа данных.
## Тип данных xsd: string используется, если лексическое значение относится к
## лексическому пространству xsd: string.
## В противном случае используется тип данных rdf: XMLLiteral.
## Лексическое значение — это содержимое
## дочернего элемента символа (если таковой присутствует); в противном случае это содержимое
## элемента cl: Name.
Name = element Name { Name.type }
Name.type |= clCommon, (cri.attrib | symbol)
Name.type |= text & anyElement*
symbol = element symbol { symbol.type }
symbol.type = text & anyElement*
## Ограниченные имена в кванторах
## Элемент cl: Name, содержащий один или несколько дочерних элементов cl: type,
## является ограниченным именем. ## У него отсутствует явный
## эквивалент в абстрактном синтаксисе.
## Квантор, чья последовательность связывания содержит
## одно или несколько ограниченных имен, является синтаксическим сахаром
## для квантора, чья последовательность связывания содержит
## только неограниченные имена и чье количественное высказывание
## является модификацией исходного количественного высказывания:
## — если ограниченное имя связано с помощью квантора всеобщности, то
## количественное высказывание заменяется импликацией, часть
## if которой является атомарным высказыванием,
## подтверждающим принадлежность обозначения имени к отношению,
## связанному с ограничивающим типом, а часть
## then — исходным количественным высказыванием;
## — если ограниченное имя связано с помощью квантора существования, то
## количественное высказывание заменяется конъюнкцией
## двух высказываний, в которой
## один конъюнкт является атомарным высказыванием,
```

```
## подтверждающим принадлежность обозначения имени к отношению,
## связанному с ограничивающим типом, а другой —
## исходным количественным высказыванием.
Quantifier.type |= clCommon, (Name | Name-constrained)*, clSentence
Name-constrained = element Name { Name-constrained.type }
Name-constrained.type = clCommon, type+, (cri.attrib | symbol)
type = element type { type.type }
type.type = Name
## -Данные
## Элемент cl: Data представляет собой имя (V) абстрактного синтаксиса,
## приведенного в фиксированной интерпретации.
## Фиксированная интерпретация указывается в соответствии
## с сопоставлением типа данных с лексическим значением.
## Лексическое значение — это содержимое
## дочернего элемента символа (если таковой присутствует); в противном случае это содержимое
## элемента cl: Data.
## Если атрибут типа данных присутствует, то значение атрибута
## в виде IRI определяется сопоставлением типа данных.
## Если атрибут типа данных отсутствует, а
## лексическое значение относится к лексическому пространству xsd: string,
## используется тип данных xsd: string.
## В противном случае используется тип данных rdf: XMLLiteral.
## Если лексическое значение не относится к лексическому пространству
## указанного типа данных, это является синтаксической ошибкой, даже если
## документ XML действителен в соответствии с этой схемой.
clTerm |= Data
Data = element Data { Data.type }
Data.type |= clCommon, symbol-data
Data.type |= datatype.attrib?, (text & anyElement*)
symbol-data = element symbol { symbol-data.type }
symbol-data.type = datatype.attrib?, (text & anyElement*)
datatype.attrib = attribute datatype { curieOrAbsIRI.datatype }
## Тип данных
clStatement |= Datatype
Datatype = element Datatype{ Datatype.type }
Datatype.type |= clCommentable, cri.attrib, xsdUserDefined.type
xsdUserDefined.type|= element xs: simpleType { xsdSimpleType.type}
xsdSimpleType.type |= xsdSimpleDerivation
xsdSimpleDerivation |= xsdRestriction
xsdSimpleDerivation |= xsdList
xsdSimpleDerivation |= xsdUnion
xsdRestriction |= element xs: restriction { baseAtt?, xsdSimpleRestrictionModel}
baseAtt = attribute base { xsd: QName}
xsdSimpleRestrictionModel |= element xs: simpleType { xsdSimpleType.type }?, xsdFacets*
```

```
xsdFacets |= xsdMinExclusive
xsdFacets |= xsdMinInclusive
xsdFacets |= xsdMaxExclusive
xsdFacets |= xsdMaxInclusive
xsdFacets |= xsdTotalDigits
xsdFacets |= xsdFractionDigits
xsdFacets |= xsdLength
xsdFacets |= xsdMinLength
xsdFacets |= xsdMaxLength
xsdFacets |= xsdEnumeration
xsdFacets |= xsdWhiteSpace
xsdFacets |= xsdPattern
xsdFacet |= attribute value { xsd: string}
xsdMinExclusive |= element xs: minExclusive { xsdFacet }
xsdMinInclusive |= element xs: maxExclusive { xsdFacet }
xsdMaxExclusive |= element xs: minInclusive { xsdFacet }
xsdMaxInclusive |= element xs: maxInclusive { xsdFacet }
xsdNumFacet |= attribute value { xsd: nonNegativeInteger }
xsdPosNumFacet |= attribute value { xsd: positiveInteger }
xsdTotalDigits |= element xs: totalDigits { xsdPosNumFacet }
xsdFractionDigits |= element xs: fractionDigits { xsdNumFacet }
xsdLength |= element xs: length { xsdNumFacet }
xsdMinLength |= element xs: minLength { xsdNumFacet }
xsdMaxLength |= element xs: maxLength { xsdNumFacet }
xsdEnumeration |= element xs: enumeration { xsdFacet }
xsdWhiteSpace |= element xs: whiteSpace {
attribute value { xsd: NMTOKEN «preserve» | «replace» | «collapse» }
}
xsdPattern |= element xs: pattern { xsdFacet }
xsdList = element xs: list {
attribute itemType { xsd: QName} |
element xs: simpleType { xsdSimpleType.type}
}
xsdUnion = element xs: union {
attribute memberTypes { xsd: string} |
element xs: simpleType { xsdSimpleType.type}+
}
## Маркеры последовательности
## Элемент cl: Marker представляет собой маркер последовательности (SMark)
## абстрактного синтаксиса. ## Фактические символ маркера последовательности,
т. е. сущности, которая относится к лексикону,
## создается путем сопоставления лексического
## значения элемента cl: Marker с пространством значений
## типа данных.
```

```
## Тип данных xsd: string используется, если лексическое значение относится к
## лексическому пространству xsd: string.
## В противном случае используется тип данных rdf: XMLLiteral.
## Лексическое значение — это содержимое
## дочернего элемента символа (если таковой присутствует); в противном случае это содержимое
## элемента cl: Marker.
Marker = element Marker { Marker.type }
Marker.type |= clCommon, symbol
Marker.type |= text & anyElement*
## Шаблоны clCommon и clCommentable:
## Prefixes, Comments, Keys
clCommentable = clCommon, label?, Comment*
clCommon = base.attrib?, Prefix*
## Элемент cl: Prefix является синтаксическим механизмом для
## сокращения IRI. Он не соответствует явным
## частям абстрактного синтаксиса.
## Перед сопоставлением с абстрактным синтаксисом все CURIE,
## которые выступают в качестве значений атрибутов XCL,
## должны быть расширены до IRI в соответствии с определениями префиксов.
## Определения пространства имен XML не следует использовать для определения
## префиксов CURIE.
## Определения префиксов используются внутри родительского элемента
## и его дочерних элементов.
## При наличии противоречивых определений префиксов:
## определения префиксов, которые являются дочерними по отношению к элементу E, имеют приоритет
## над теми, что являются дочерними по отношению к предшественникам элемента E;
## определения префиксов имеют приоритет перед предшествующими аналогами.
Prefix = element Prefix { Prefix.type }
Prefix.type = pre.attrib, iri.attrib
pre.attrib = attribute pre { pre.datatype }
pre.datatype = xsd: NCName?
## Элемент cl: Comment представляет собой комментарий абстрактного синтаксиса,
## Лексическое значение элемента cl: Comment является его содержимым.
## Атрибуты могут использоваться для указания типа данных, который будет применен при сопоставлении
## лексического значения с пространством значений типа данных.
## Благодаря этому комментарий абстрактного синтаксиса рассматривается как «фрагмент данных».
Comment =
element Comment {
attribute * { text }*,
(text & anyElement*)
}
label = key.attrib
iri.attrib = attribute iri { absIRI.datatype }
cri.attrib = attribute cri { curieOrAbsIRI.datatype } key.attrib
```

```
= attribute key { curieOrAbsIRI.datatype }
curieOrAbsIRI.datatype = curie.datatype | absIRI.datatype
curie.datatype = xsd: string { minLength = “1” pattern = “(([\i-[:]][\c-[:]]*)?:)?.*” }
absIRI.datatype = xsd: anyURI { pattern = “[\i-[:]][\c-[:]]*:.*” }
anyElement =
element * { attribute *
{ text }*, (text &
anyElement*)
}
base.attrib = attribute xml: base {xsd: anyURI}
```

В.3 Семантика XCL

CL можно напрямую преобразовать в XCL в форме, которая будет отображать синтаксис CL непосредственно в разметке XML. Поскольку XML не имеет внутренней семантики, назначение Б.2 состоит в том, чтобы присвоить каждой функции в абстрактном синтаксисе (см. 6.1) соответствующую конструкцию на синтаксисе XCL, которая несет в себе семантику, указанную в 6.2. В документации схемы четко указаны эти элементы.

Кроме того, существуют элементы XCL, которые являются синтаксическим сахаром для более сложного выражения абстрактного синтаксиса.

Другие семантики не подразумеваются и не выражаются.

В.4 Совместимость с XCL

По следующим причинам XCL является строго совместимым диалектом. Реальный синтаксис XCL из Б.2 точно соответствует абстрактному синтаксису CL, описанному в 6.1. Более того, поскольку семантика XCL (обобщенная в Б.3) скопирована с семантики CL, описанной в 6.2, она синтаксически и семантически соответствует основному тексту настоящего стандарта.

Приложение Г
(справочное)

Перевод между диалектами

Перевод — это сопоставление *tr* из выражений в тексте на определенном диалекте A (исходный диалект) с выражениями в тексте на определенном диалекте B (диалект перевода). При этом для каждой интерпретации *I* словаря текста на диалекте A должна существовать интерпретация *J* словаря текста на диалекте B, а для каждой интерпретации *J* словаря текста на диалекте B — интерпретация *I* словаря текста на диалекте A, где *I*(E)=*J*(*tr*(E)) для любого выражения E в тексте на диалекте A, а *tr* — это перевод. Поскольку все диалекты CL имеют одинаковые условия истинности, перевод, как правило, довольно прямолинеен. Однако при переводе между классическими и неклассическими диалектами возникают сложности.

Перевод с классического диалекта A на неклассический диалект B должен указывать, какие термины диалекта A не являются дискурсивными. Для каждого имени на неклассическом диалекте, которое обозначает предмет в области, перевод должен вводить *дискурсивное имя*, расширение которого на диалекте B является областью интерпретации A. Кроме того, перевод должен ограничивать все кванторы в тексте до диапазона этой области и гарантировать, что недискурсивные имена на классическом диалекте обозначают сущности за пределами этой области. Другого перевода не требуется. Конструкция области содержит универсальную технику для таких переводов: текст на диалекте A имеет то же значение, что и текст на диалекте B с именем области и недискурсивными именами из текста, перечисленными в списке исключений.

При переводе с неклассического диалекта B на диалект A с дискурсивными пресуппозициями имена должны использоваться с учетом ограничений диалекта. Для этого к переводам может понадобиться добавить аксиомы. Это необходимо, чтобы гарантировать, что область интерпретации классического перевода любого текста будет соответствовать исходной вселенной интерпретации неклассического текста. Существует общий метод (*перевод holds-app*) для перевода любого диалекта CL на аналогичный классический диалект. Предполагается наличие предиката *holds* и оператора *app*, которые отсутствуют в словарях. В частности (для неклассических диалектов), атомарное высказывание с предикатом P и последовательностью аргументов $S_1 \ldots S_n$ трансформируется в атомарное высказывание с предикатом *holds* и последовательностью аргументов P $S_1 \ldots S_n$. Термин с оператором O и последовательностью аргументов $S_1 \ldots S_n$ преобразуется в термин с оператором *app* и последовательностью аргументов O $S_1 \ldots S_n$. Введенный предикат и оператор не требуют других аксиом. Их единственная роль заключается в том, чтобы позволить операторам и предикатам диалекта B обозначать сущности в области перевода диалекта A.

Некоторые диалекты налагают различные ограничения на обозначения, например, чтобы связанные имена имели определенную лексическую форму или операторы и предикаты использовались с последовательностью аргументов определенной длины (обычно называемой *арностью* оператора или предиката). Перевод на диалект с такими ограничениями обычно выполняется путем переписывания имен в соответствии с имеющимися ограничениями и устранения возможной «игры слов» при упоминании имени в диалекте перевода (например, путем добавления суффиксов). В этих случаях также может потребоваться ввести отдельные предикаты и операторы *hold-n* и *app-n* для каждой арности. Приложения, от которых требуется точный перевод нескольких текстов, должны обеспечивать согласованность между многократными упоминаниями имен.

Библиография

[1] Genesereth M.R., & Fikes R.E. «Справочное руководство по формату обмена знаниями версии 3.0». Кафедра компьютерных наук, Стэнфордский университет, технический отчет KSL-92-86, июнь 1992 г. («Knowledge Interchange Format Version 3.0 Reference Manual». Computer Science Department, Stanford University, Technical Report KSL-92-86, June 1992)

[2] ISO/IEC 14977:1996, Информационные технологии. Синтаксический метаязык. Расширенная форма (Information technology — Syntactic metalanguage — Extended BNF)

[3] Sowa J.F. «Концептуальные структуры: обработка информации с помощью разума и машины». Издательство Эдисон-Уэсли, г. Ридинг, Массачусетс, 1984 г. (Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, Mass., 1984)

[4] Dürst M., & Suignard M. «Интернационализированные идентификаторы ресурсов (IRI)». Общество Интернета, 2005 г.). http: //www .ietf .org/rfc/rfc3987.txt («Internationalized Resource Identifiers (IRIs)», The Internet Society, 2005)

[5] Mealling M., & Denenberg R. RFC 3305 — Отчет объединенной стратегической группы W3C/IETF URI: унифицированные идентификаторы ресурсов (URI), URL-адреса и универсальные имена ресурсов (URN): пояснения и рекомендации,) 2002 г. («RFC 3305 — Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations»)

[6] ISO/IEC 10646:2014, Информационные технологии. Универсальный набор символов (UCS) (Information technology — Universal coded character set (UCS))

[7] Berners-Lee T., Hendler J., Lassila O. «Семантическая сеть». Научный американский журнал. (The Semantic Web. Scientific American journal, № 284, 2001 г.)

УДК 81.37; 004.423.23:006.354

ОКС 35.060,
01.040.35

Ключевые слова: общая логика CL, семантика общей логики, абстрактный синтаксис

Редактор *Л.С. Зимилова*
Технический редактор *В.Н. Прусакова*
Корректор *О.В. Лазарева*
Компьютерная верстка *Л.А. Круговой*

Сдано в набор 28.10.2021. Подписано в печать 30.11.2021. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 7,44. Уч.-изд. л. 5,92.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении в ФГБУ «РСТ»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru