Документация scl-machine

Реализация scl-машины
:= [Программный вариант реализации машины логического вывода scl]
:= [Машина логического вывода scl]
:= [scl-машина]
:= [scl-machine]
:= [ostis-inference]
∈ *машина обработки знаний*
⇐ *программная модель**:
Абстрактная scl-машина
⇒ *внутренний язык**:
Язык SCL
⇒ *декомпозиция программной системы**:
{• *База знаний scl-machine*
• *Решатель задач scl-machine*
• *Интерфейс scl-machine*
}

Решатель задач scl-machine
⇒ *обобщённая декомпозиция**:
{• *Агент прямого логического вывода*
• *Агент обратного логического вывода*
⇒ *примечание**:
[Не реализовано.]
• *Агент применения правил вывода*
⇒ *примечание**:
[Не реализовано.]
• *Агент эквивалентных преобразований логической формулы*
⇒ *примечание**:
[Не реализовано.]
⇒ *реализованные логические связки**:
{• *импликация**
• *дизъюнкция**
• *конъюнкция**
• *отрицание**
}
⇒ *не реализованные логические связки**:
{• *эквиваленция**
• *строгая дизъюнкция**
}
}

менеджер логического вывода
:= [InferenceManagerAbstract]
⇒ *примечание**:
[менеджер логического вывода определяет, каким образом производиться обход и применение логических формул.]
⇒ *программный интерфейс**:
Программный интерфейс менеджера логического вывода
⇒ *обязательные понятия для спецификации заданной сущности**:
{• *искатель атомарных логических формул*
:= [TemplateSearcherAbstract]
• *менеджер обработки атомарных логических формул*
:= [TemplateManagerAbstract]
• *менеджер дерева решений*
:= [SolutionTreeManagerAbstract]
}
⇒ *декомпозиция**:
{• *менеджер прямого логического вывода по цели*
:= [DirectInferenceManagerTarget]
• *менеджер прямого логического вывода по всем логических формулам*

:= [DirectInferenceManagerAll]
}

Программный интерфейс менеджера логического вывода
∋ *Метод применения логического вывода*
⇒ *заголовок метода**:
[virtual bool applyInference(InferenceParamsConfig const & inferenceParamsConfig) = 0;]
⇒ *примечание**:
[Главный метод менеджера логического вывода, который определяет порядок обхода и формул.]
∋ *Метод применения логической формулы*
⇒ *заголовок метода**:
[LogicFormulaResult useFormula(ScAddr const & formula, ScAddr const & outputStructure);]
⇒ *примечание**:
[Метод менеджера логического вывода, который анализирует логическую формулу и генерирует атомарные логические формулы по импликации.]

искатель атомарных логических формул
⇒ *программный интерфейс**:
Программный интерфейс искателя атомарных логических формул
= {• *метод поиска атомарных логических формул по параметрам*
⇒ *заголовок метода**:
[virtual void searchTemplate(ScAddr const & templateAddr, ScTemplateParams const & templateParams, std::set<std::string> const & varNames, Replacements & result) = 0;]
⇒ *примечание**:
[Метод ищет конструкции в базе знаний по графу-образцу (логической атомарной формулы) с учётом переданных параметров графа-образца и создаёт соответствие между sc-переменными формулы и соответствующими ей константными sc-элементами.]
• *метод поиска атомарных логических формул по множеству параметров*
⇒ *заголовок метода**:
[virtual void searchTemplate(ScAddr const & templateAddr, vector<ScTemplateParams> const & scTemplateParamsVector, std::set<std::string> const & varNames, Replacements & result);]
⇒ *примечание**:
[Метод вызывает ***метод поиска атомарных логических формул по параметрам*** в цикле для переданного множества параметров поиска.]
}
⇒ *декомпозиция**:
{• *искатель атомарных логических формул по всей базе знаний*
:= [TemplateSearcher]
⇒ *примечание**:
[Поиск конструкций осуществляется по всей базе знаний.]
• *искатель атомарных логических формул в структурах*
:= [TemplateSearcherInStructures]
⇒ *примечание**:
[Все найденные конструкции должны принадлежать любой структуре из множества входных структур.]
}

менеджер обработки атомарных логических формул
⇒ *программный интерфейс**:
Программный интерфейс менеджера обработки атомарных логических формул
= {• *метод создания параметров поиска атомарной логической формулы*
⇒ *заголовок метода**:
[virtual std::vector<ScTemplateParams> createTemplateParams(ScAddr const & scTemplate) = 0;]
⇒ *примечание**:

[Метод формирует множество параметров атомарной логической формулы.]
}
⇒ *декомпозиция**:
{• *менеджер обработки атомарных логических формул*
:= [TemplateManager]
⇒ *примечание**:
[Формирование параметров осуществляется по всей базе знаний. Происходит поиск переменных sc-узлов в атомарной логической формулы с их классами и формируется соответствие их с константными sc-узлами с такими же классами в базе знаний.]
• *менеджер обработки атомарных логических формул с фиксированными аргументами*
:= [TemplateManagerFixedArguments]
⇒ *примечание**:
[Формирование параметров осуществляется по переданным аргументами и спецификации формулы. Переменная, формуле под первой ролью, соответствует первому аргументу из множества аргументов логического вывода.]
}

менеджер дерева решений
⇒ *программный интерфейс**:
Программный интерфейс менеджера дерева решений
= {• *метод создания узла дерева решения*
⇒ *заголовок метода**:
[virtual bool addNode(ScAddr const & formula, Replacements const & replacements) = 0;]
⇒ *примечание**:
[Данный метод определяет структуру и создание узлов дерева решения.]
}
⇒ *декомпозиция**:
{• *менеджер дерева решений с подстановками*
:= [SolutionTreeManager]
⇒ *примечание**:
[Узел такого дерева решения состоит из применённой логической формулы и соответствий sc-переменных sc-константам, которые были использованы в атомарных формулах.]
• *пустой менеджер дерева решений*
:= [SolutionTreeManagerEmpty]
⇒ *примечание**:
[В такой реализации менеджера дерева решений узлы не создаются. Такая реализация сделана из соображений оптимизации.]
}

конфиг менеджера логического вывода
:= [InferenceConfig]
⇒ *декомпозиция**:
{• *Конфиг процесса логического вывода*
:= [InferenceFlowConfig]
⇒ *примечание**:
[Такой конфиг используется при создании менеджера логического вывода.]
∋ *generateOnlyUnique*
⇒ *примечание**:
[Определяет, нужно ли генерировать уже существующие конструкции в базе знаний. От этого зависит, нужно ли перед генерацией атомарной логической формулы искать её в базе знаний. Если не искать, это даёт большой прирост в производительности логического вывода.]
∋ *generateOnlyFirst*
⇒ *примечание**:
[Определяет, нужно ли прерывать генерацию атомарной логической формулы по множеству аргументов после первой успешной генерации.]
∋ *generateSolutionTree*
⇒ *примечание**:

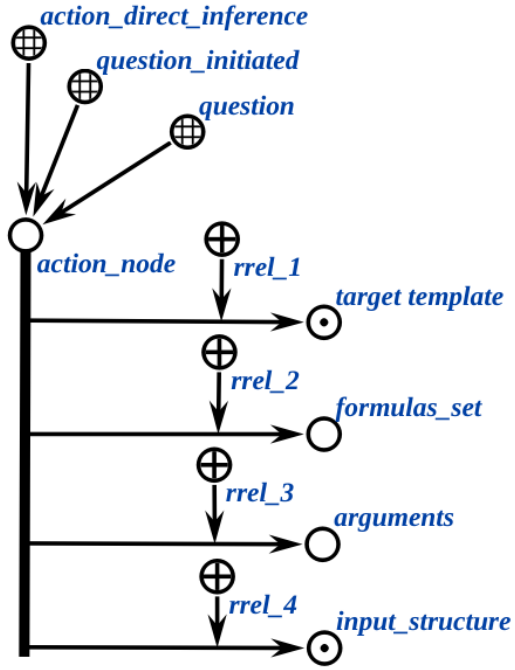
[Определяет, нужно ли создавать узлы в дереве решений. Если не нужно, то в процессе логического вывода используется ***пустой менеджер дерева решений***.]

- *Конфиг параметров логического вывода*
  - := [InferenceParamsConfig]
  - ⇒ *примечание**:
    [Такой конфиг передаётся при каждой использовании логического вывода с уже созданным ***менеджером логического вывода***.]
  - ∋ *formulasSet*
  - ∋ *arguments*
  - ∋ *inputStructures*
  - ∋ *outputStructure*
  - ∋ *targetStructure*

}

объект создания менеджера логического вывода
:= [InferenceManagerFactory]
:= [фабрика менеджера логического вывода]
⇒ *примечание**:
[С помощью него создаётся менеджер логического вывода в соответствии с переданным ***Конфигом процесса логического вывода***.]

Агент прямого логического вывода
:= [sc-агент прямого логического вывода]
⇒ *примечание**:
[Задачей sc-агента прямого логического вывода является генерация новых знаний на основе некоторых логических утверждений. Данный sc-агент активируется при появлении в sc-памяти инициированного действия, принадлежащего классу *действие прямого логического вывода*. После проверки sc-агентом условия инициирования выполняется процесс прямого логического вывода.]
⇒ *пример входной конструкции**:
[

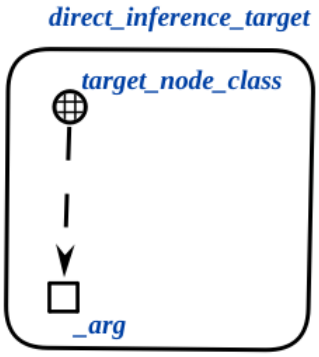


]
⇒ *аргументы агента**:
⟨• *_target_ template*
:= [targetTemplate]
:= [targetStatement]
:= [шаблон цели]
:= [ожидаемый результат выполнения логического вывода]
⇒ *пояснение**:

[Шаблон, успешный поиск которого показывает, что цель логического вывода достигнута и применение правил можно прекратить.]

⇒ *описание примера**:

[



]

- *_formulas_set*

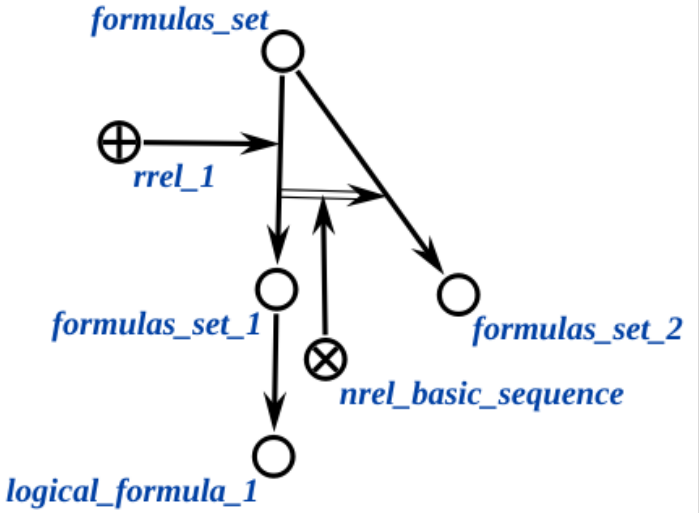
  ≔ [formulasSet]

  ⇒ *пояснение**:

  [Ориентированное множество множеств формул, применяя которые требуется совершить логический вывод. Первым элементом множества является множество формул, которые применяются в первую очередь, а каждое следующее множество формул применяется после предыдущего. Таким образом указываются приоритеты множеств формул.]

  ⇒ *описание примера**:

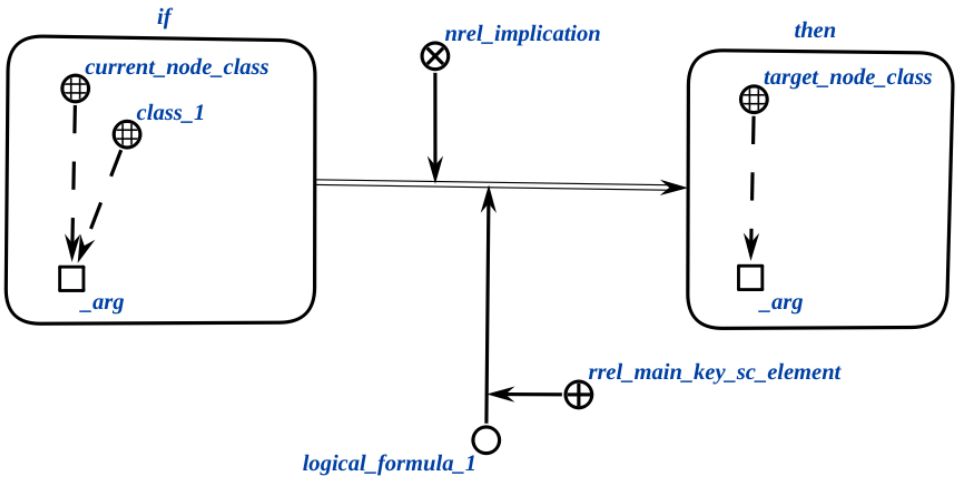
  [



  ]

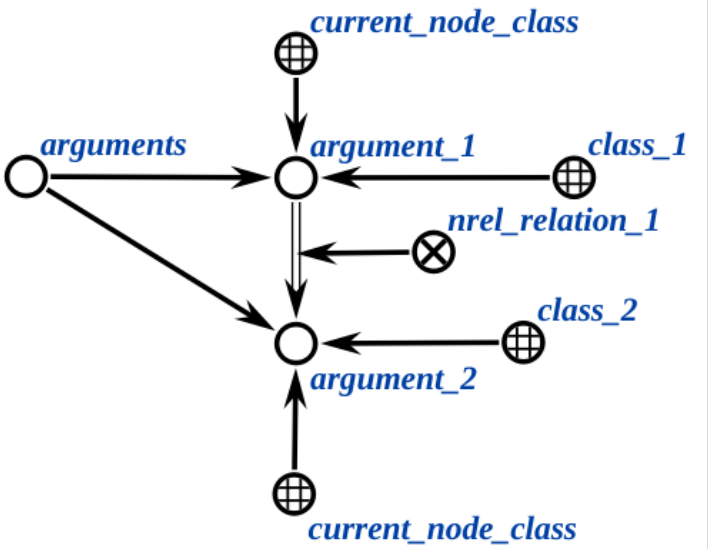
  ⇒ *описание примера**:

  [



]

- *arguments*
  ⇒ *пояснение**:
  [Множество, элементы которого используются при применении правил. Каждый sc-узел этого множества подставляется как значение переменных атомарных логических формул (в том числе шаблона цели).]
  ⇒ *примечание**:
  [Можно использовать структуру всей базы знаний системы, например, sc-узел ***База знаний IMS***.]
  ⇒ *описание примера**:
  [



]

- *_intput_structure*
  ⇒ *пояснение**:
  [Структура, в которой происходит поиск при проверке истинности атомарных логических формул.]

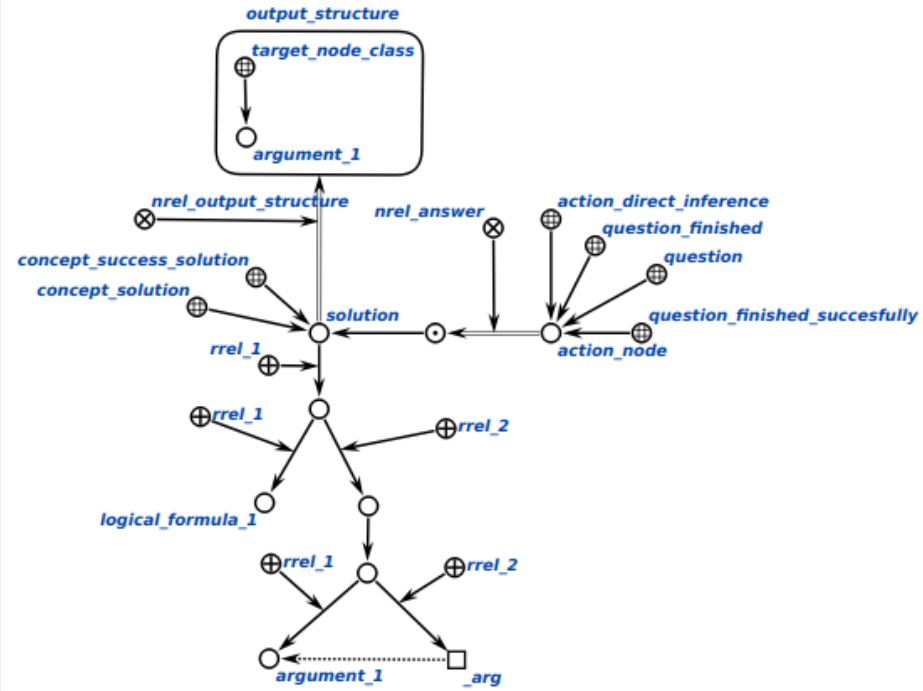
⟩

⇒ *ответ агента**:
ответ агента прямого логического выода
⇒ *примечание**:
[В результате выполнения агентом логического вывода действия, в sc-памяти формируется sc-структура, представляющая собой дерево решения. Это дерево состоит из последовательности пар, в которых первым элементом является применённое правило, а вторым - множество пар подстановок и переменных, соединённых временной дугой принадлежности, выходящей из переменного узла, при которых данное правило

выполняется. Такое дерево может быть пустым в случае, если требуемую структуру не удалось сгенерировать в ходе логического вывода. При достижении цели вывода узел дерева решения позитивно принадлежит классу concept_success_solution, при недостижении – негативно. Корень этого дерева находится в связке под отношением nrel_output_structure со структурой, в которую добавляются сгенерированные в ходе логического вывода конструкции.]

⇒ *описание примера**:

[



]

⇒ *примечание**:

[Работа агента заключается в последовательном применении правил из входного множества правил, генерируя структуры, если атомарная формула принадлежит классу формул для генерации (***concept_formula_for_generation***). Если правило применилось безуспешно, то оно добавляется во множество безуспешно применённых правил, которые применяются повторно в случае успешного применения какого-либо другого правила. Также после каждого успешного применения правила проверяется, достигнута ли цель (если она передана), и, если цель достигнута, выполнение агента завершается успешно и остальные правила не применяются.]

⇒ *обобщённый алгоритм**:

⟨• [Получение параметров агента, вызов агента;]

• [Получение всех sc-узлов из arguments, если множество валидно, заполнение ими списка аргументов;]

• [Проверка, достигнута ли уже цель в базе знаний с полученными аргументами;]

⇒ *примечание**:

[Выполняется поиск по шаблону target template с параметрами arguments. Если шаблон найден, агент завершает работу, возвращает узел, принадлежащий ***concept_success_solution***.]

• [Построение вектора очереди формул на основе множества формул. Цикл по всем правилам и пока не достигнута цель;]

⇒ *циклические операции**:

⟨• [Получение посылки логической формулы;]

• [Определение типа посылки (связка конъюнкции, дизъюнкции, отрицания или атомарная логическая формула);]

• [Проверка истинности посылки в зависимости от её типа;]

⇒ *замечание**:

[Конъюнкция, дизъюнкция, отрицание работают нестабильно.]

• [Генерация по шаблону следствия;]

• [Добавление в дерево решений узла формулы.]

⇒ *примечание**:
[Смотрите пример ответа агента.]
⟩
- [Формирование дерева применённых формул.]

⟩
⇒ *недостатки текущего состояния**:
{
- [В текущем состоянии не реализован механизм применения правил вывода, вместо него указываются формулы для генерации, используя класс ***concept_formula_for_generation***.]
- [Генерируются только атомарные формулы.]
- [Логическая связка отрицания некорректно работает с подстановками.]
- [В структуру ответа агента входит только узел solution, а не вся структура решения.]
- [Не реализована логика для входной структуры, поиск осуществляется по всей базе знаний.]

}
⇒ *преимущества текущего состояния**:
{
- [Агент работает корректно при передаче параметров в соответствии с предыдущим вариантом его реализации.]
- [Проверка входных параметров не только по невалидности sc-узла, но и проверка на непустое множество.]

}