

# Документация SCL-machine

**Программный вариант реализации логической машины интерпретации логических sc-моделей компьютерных систем**

```

:= [SCL-machine]
:= [ostis-inference]
⇒ декомпозиция программной системы*:
{
• База знаний SCL-machine
• Решатель задач SCL-machine
• Интерфейс SCL-machine
}
⇒ реализованные логические связи*:
{
• импликация*
• дизъюнкция*
• конъюнкция*
• отрицание*
}
⇒ не реализованные логические связи*:
{
• эквиваленция*
• строгая дизъюнкция*
}

```

**Решатель задач SCL-machine**

```

⇒ обобщённая декомпозиция*:
{
• Агент прямого логического вывода
• Агент обратного логического вывода
⇒ примечание*:
[Не реализовано.]
}

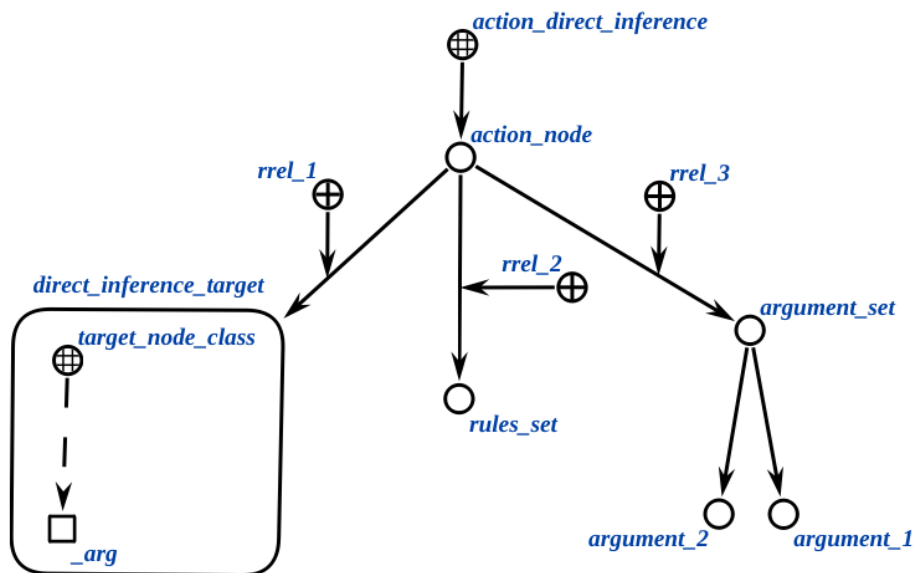
```

**Агент прямого логического вывода**

```

⇒ пример входной конструкции*:
[

```

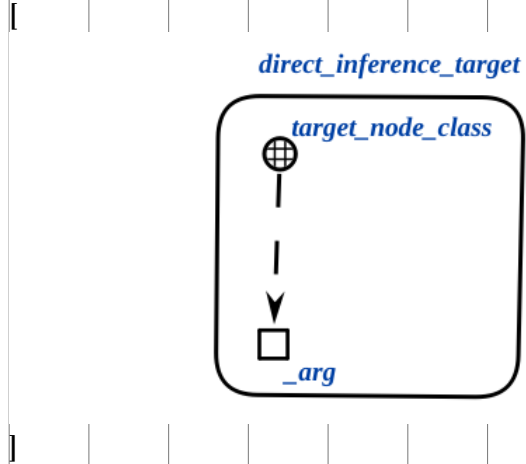


```

]
⇒ аргументы агента*:
<• _target_template
:= [targetTemplate]
:= [targetStatement]
:= [ожидаемый результат выполнения логического вывода]
⇒ пояснение*:
[Шаблон, успешный поиск которого показывает, что цель логического вывода
достигнута и применение правил можно прекратить.]

```

⇒ описание примера\*:



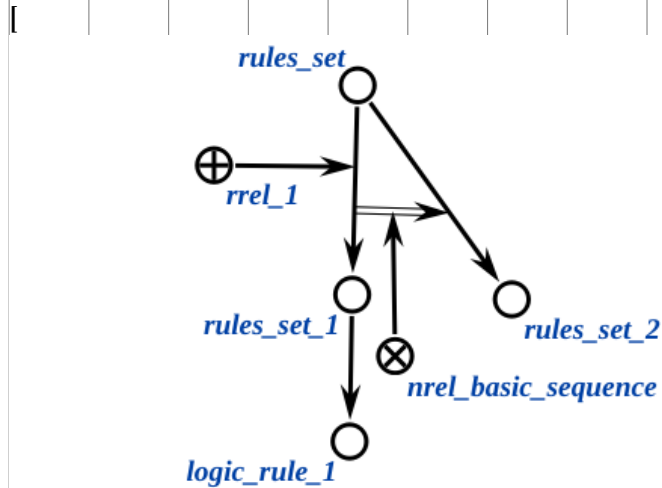
• *rule* *set*

:= [ruleSet]

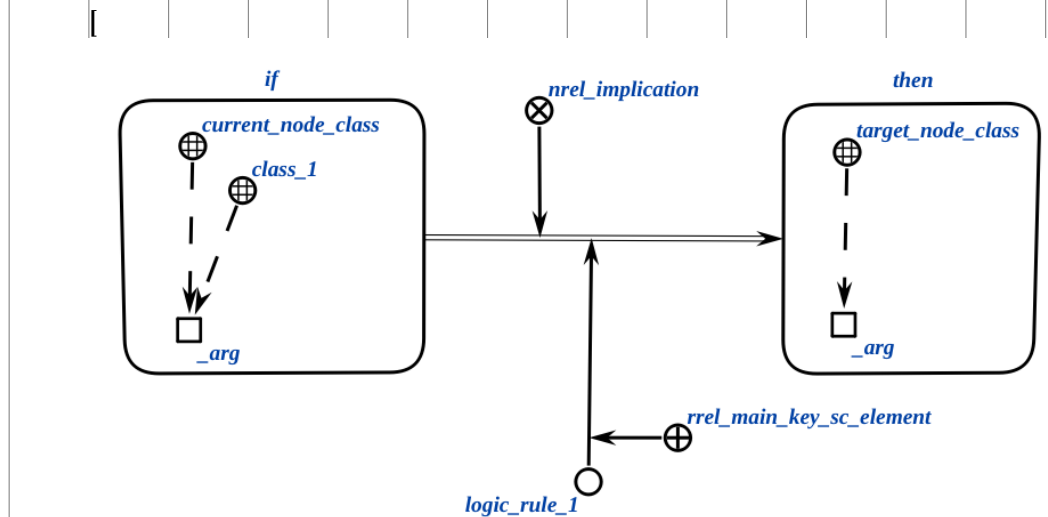
⇒ пояснение\*:

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

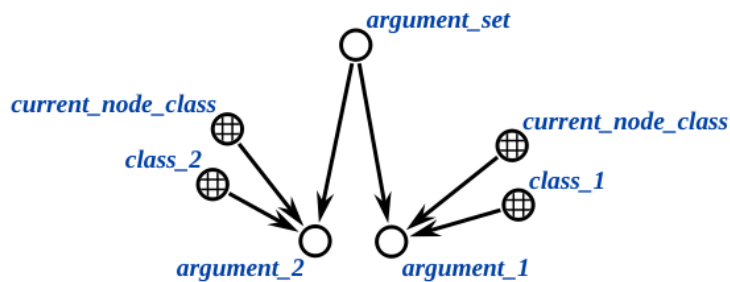
⇒ описание примера\*:



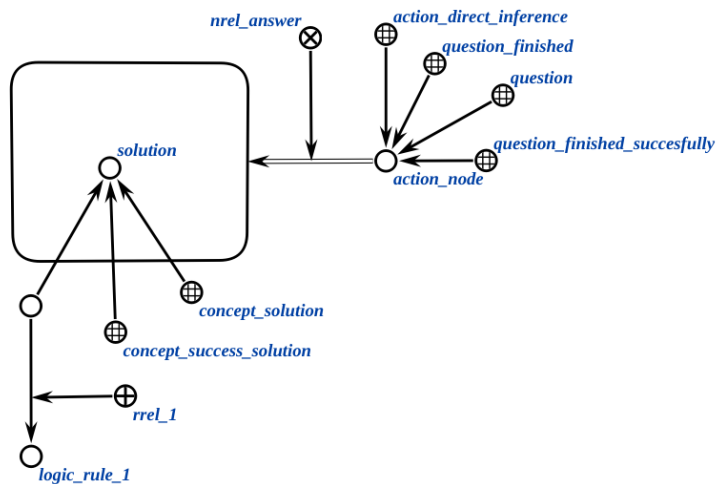
⇒ описание примера\*:



- `_argument_set`  
`:= [argumentSet]`  
 $\Rightarrow$  *пояснение\**:  
 [Множество тех элементов, которые должны быть подставлены как значение переменных шаблона цели.]  
 $\Rightarrow$  *пояснение\**:  
 [В данном примере значением переменной `_x` в шаблоне цели может быть sc-узел `argument_1`, тогда значением `_y` будет `argument_2`. Также значением переменной `_x` может быть sc-узел `argument_2`, а значением `_y` – `argument_1`.]  
 $\Rightarrow$  *описание примера\**:  
 [



- $\Rightarrow$  *пример выходной конструкции\**:  
 [



- $\Rightarrow$  *обобщённый алгоритм\**:  
 {
 • [Получение параметров агента, проверка их валидности. Вызов агента;]  
 • [Проверка, достигнута ли уже цель в базе знаний;]  
 $\Rightarrow$  *примечание\**:  
 [Выполняется поиск по шаблону `target template` с параметрами шаблона `arguments set`.]  
 • [Построение вектора очереди правил на основе множества правил. Цикл по всем правилам и пока не достигнута цель;]  
 $\Rightarrow$  *циклические операции\**:  
 {
 • [Получение посылки логического правила;]  
 • [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 правила.]

⇒ *примечание\**:

[Смотрите пример выходной конструкции.]

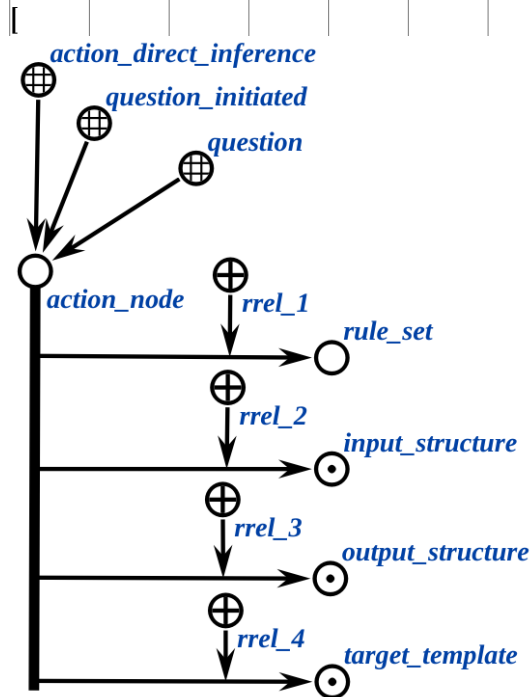
}

- [Возврат дерева применённых правил.]

}

**Агент прямого логического вывода**

⇒ *пример входной конструкции\**:



⇒ *аргументы агента\**:

⟨ • *\_rule\_set*

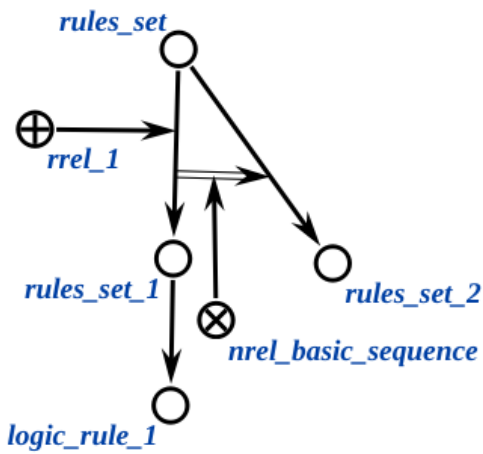
:= [ruleSet]

⇒ *пояснение\**:

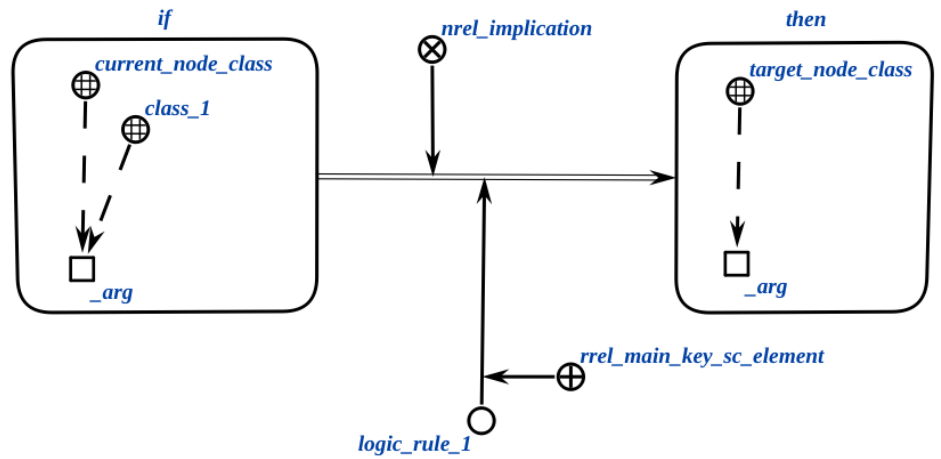
[Ориентированное множество множеств правил, применяя которые требуется совершить логический вывод. Первым элементом множества является множество правил, которые применяются в первую очередь, а каждое следующее множество правил применяется после предыдущего. Таким образом указываются приоритеты множеств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

⇒ *описание примера\**:

[



⇒ описание примера\*:  
[



• `_input_structure`

:= [inputStructure]

⇒ пояснение\*:

[Структура, элементы которой используются при применении правил. Каждый sc-узел этой структуры добавляется во множество аргументов, которые должны быть подставлены как значение переменных шаблона цели.]

⇒ аналог\*:

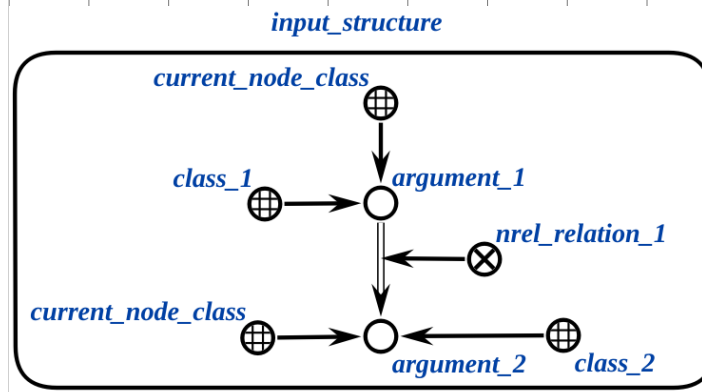
`_argument_set`

⇒ примечание\*:

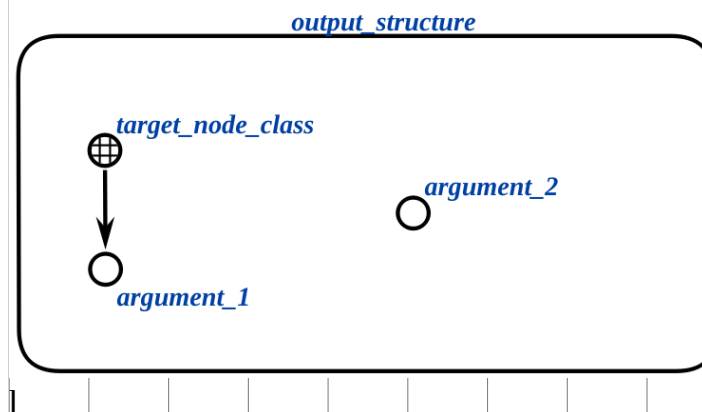
можно использовать структуру всей базы знаний системы, например, sc-узел **База знаний IMS**.

⇒ описание примера\*:

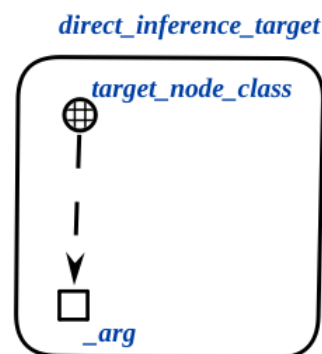
[



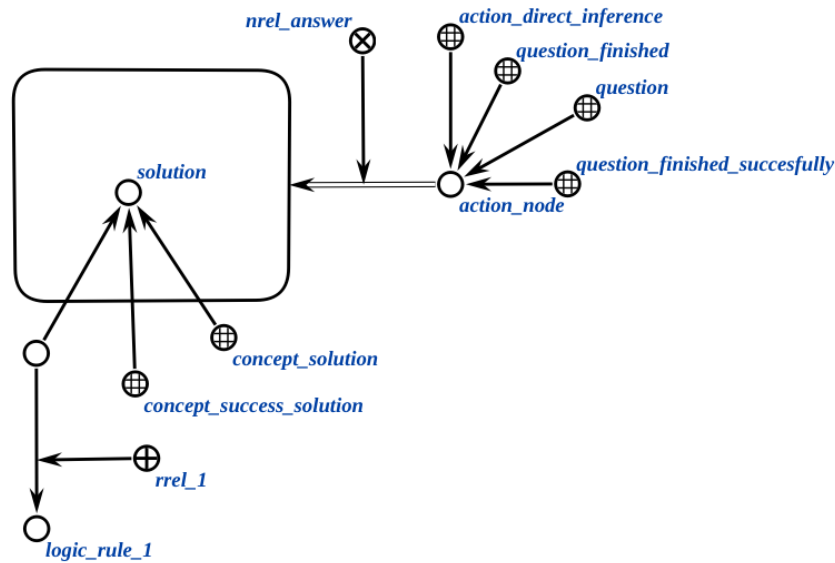
- *\_output\_structure*  
 ::= [outputStructure]  
 ⇒ *пояснение\**:  
 [Структура, в которую добавляются сгенерированные в ходе логического вывода конструкции.]  
 ⇒ *примечание\**:  
 [можно использовать структуру всей базы знаний системы, например, sc-узел **База знаний IMS**.]  
 ⇒ *описание примера\**:  
 [



- *\_target\_template*  
 ::= [targetTemplate]  
 ::= [targetStatement]  
 ::= [ожидаемый результат выполнения логического вывода]  
 ⇒ *пояснение\**:  
 [Шаблон, успешный поиск которого показывает, что цель логического вывода достигнута и применение правил можно прекратить.]  
 ⇒ *описание примера\**:  
 [



⇒ пример выходной конструкции\*:



⇒ 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 успешно, то создаётся отношение *nrel\_satisfiable\_formula* между правилом и моделью, на которой это правило выполнимо. 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 цель передалась), 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 inputStructure, если структура валидна, заполнение ими списка аргументов;]
- [Проверка, достигнута ли уже цель в базе знаний с полученными аргументами;]

⇒ примечание\*:

[Выполняется поиск по шаблону target template с параметрами структуры input\_structure. Если шаблон найден, агент завершает работу, возвращает узел, принадлежащий *concept\_success\_solution*.]

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

⇒ циклические операции\*:

- [Получение посылки логического правила;]
  - [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 правила.]
- ⇒ примечание\*:
- [Смотрите пример выходной конструкции.]

- [Возврат дерева применённых правил.]

⇒ недостатки текущего состояния\*:

{•



[Импликация в текущем состоянии интерпретируется не как логическая связка, а как отношение **выводимости\***, то есть импликация не возвращает логическую константу, а генерирует новые знания. Вместо этого должны использоваться правила вывода, например, Modus ponens и другие правила, в процессе логического вывода.]

- [В структуру ответа агента входит только узел solution, а не вся структура решения.]
- [Нет реализации вызова агента в зависимости от переданных аргументов (старый или новый интерфейс).]

}

⇒

*преимущества текущего состояния\*:*

- [Проверка передаваемых параметров]

⇒ *примечание\**:

[Если не передано множество правил, успешность завершения агента зависит от того, была ли достигнута цель.]

⇒ *примечание\**:

[Если не передана входная структура, используется База знаний IMS.]

⇒ *примечание\**:

[Если не передана выходная структура, результат генерации никуда не записывается.]

⇒ *примечание\**:

[Если не передан шаблон цели, агент проверяет выполнимость каждого правила на входной структуре.]

- [Сохранён интерфейс предыдущей реализации агента с тремя аргументами.]

}