GSV evaluator library

0.1

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 iif_sadaf Namespace Reference

Namespaces

- namespace talk

4.2 iif_sadaf::talk Namespace Reference

Namespaces

- namespace GSV

4.3 iif_sadaf::talk::GSV Namespace Reference

Classes

- struct Evaluator

  Represents an evaluator for logical expressions.
- struct IModel

  Interface for class representing a model for QML without accessiblity.
- struct InformationState

  Represents an information state based on a given model.
- struct Possibility

  Represents a possibility as understood in the underlying semantics.
- struct ReferentSystem

  Represents a referent system for variable assignments.

Functions

- InformationState update (const InformationState &input_state, std::string_view variable, int individual)

  Updates the information state with a new variable-individual assignment.
- bool extends (const InformationState &s2, const InformationState &s1)

  Determines if one information state extends another.
- std::string str (const InformationState &state)
- bool isDescendantOf (const Possibility &p2, const Possibility &p1, const InformationState &s)

  Determines if one possibility is a descendant of another within an information state.
- bool subsistsIn (const Possibility &p, const InformationState &s)

  Checks if a possibility subsists in an information state.
- bool subsistsIn (const InformationState &s1, const InformationState &s2)

  Checks if an information state subsists within another.
- bool extends (const Possibility &p2, const Possibility &p1)

  Determines whether one Possibility extends another.
- bool operator< (const Possibility &p1, const Possibility &p2)
- std::string str (const Possibility &p)
- bool extends (const ReferentSystem &r2, const ReferentSystem &r1)

  Determines whether one ReferentSystem extends another.
- std::string str (const ReferentSystem &r)

  Represents a referent system for variable assignments.
- int termDenotation (std::string_view term, int w, const IModel &m)

  Retrieves the denotation of a term in a given world within a model.
- const std::set< std::vector< int > > & predicateDenotation (std::string_view predicate, int w, const IModel &m)

  Retrieves the denotation of a predicate in a given world within a model.
- int variableDenotation (std::string_view variable, const Possibility &p)

  Retrieves the denotation of a variable in a given possibility.

4.3.1 Function Documentation

4.3.1.1 extends() [1/3]

```
bool iif_sadaf::talk::GSV::extends (
            const InformationState & s2,
            const InformationState & s1)
```

Determines if one information state extends another.

Checks whether every possibility in s2 extends at least one possibility in s1.

Parameters

s2	The potentially extending information state.
s1	The base information state.

Returns

True if s2 extends s1, false otherwise.

Definition at line 142 of file information_state.cpp.

4.3.1.2 extends() [2/3]

```
bool iif_sadaf::talk::GSV::extends (
            const Possibility & p2,
            const Possibility & p1)
```

Determines whether one Possibility extends another.

A Possibility `p2` extends `p1` if:

- They have the same world.
- Every peg mapped in `p1` has the same individual in `p2`.

Parameters

p2	The potential extending Possibility.
p1	The base Possibility.

Returns

True if `p2` extends `p1`, false otherwise.

Definition at line 64 of file possibility.cpp.

4.3.1.3 extends() [3/3]

```
bool iif_sadaf::talk::GSV::extends (
            const ReferentSystem & r2,
            const ReferentSystem & r1)
```

Determines whether one ReferentSystem extends another.

This function checks whether the referent system `r2` extends the referent system `r1`. A referent system `r2` extends `r1` if:

- The range of `r1` is a subset of the range of `r2`.
- The domain of `r1` is a subset of the domain of `r2`.
- Variables in `r1` retain their values in `r2`, or their values are new relative to `r1`.
- New variables in `r2` have new values relative to `r1`.

Parameters

r2	The potential extending ReferentSystem.
r1	The base ReferentSystem.

Returns

True if `r2` extends `r1`, false otherwise.

Definition at line 96 of file referent_system.cpp.

4.3.1.4 isDescendantOf()

```
bool iif_sadaf::talk::GSV::isDescendantOf (
            const Possibility & p2,
            const Possibility & p1,
            const InformationState & s)
```

Determines if one possibility is a descendant of another within an information state.

A possibility p2 is a descendant of p1 if it extends p1 and is contained in the given information state.

Parameters

p2	The potential descendant possibility.
p1	The potential ancestor possibility.
s	The information state in which the relationship is checked.

Returns

True if p2 is a descendant of p1 in s, false otherwise.

Definition at line 183 of file information_state.cpp.

4.3.1.5 operator<()

```
bool iif_sadaf::talk::GSV::operator< (
            const Possibility & p1,
            const Possibility & p2)
```

Definition at line 76 of file possibility.cpp.

4.3.1.6 predicateDenotation()

```
const std::set< std::vector< int > > & iif_sadaf::talk::GSV::predicateDenotation (
            std::string_view predicate,
            int w,
            const IModel & m)
```

Retrieves the denotation of a predicate in a given world within a model.

Parameters

predicate	The predicate to be interpreted.
w	The world in which the predicate is interpreted.
m	The model containing the interpretation.

Returns

The set of tuples representing the predicate's interpretation.

Exceptions

std::out_of_range	if the predicate does not exist.

Definition at line 28 of file semantic_relations.cpp.

4.3.1.7 str() [1/3]

```
std::string iif_sadaf::talk::GSV::str (
            const InformationState & state)
```

Definition at line 158 of file information_state.cpp.

4.3.1.8 str() [2/3]

```
std::string iif_sadaf::talk::GSV::str (
            const Possibility & p)
```

Definition at line 81 of file possibility.cpp.

4.3.1.9 str() [3/3]

```
std::string iif_sadaf::talk::GSV::str (
            const ReferentSystem & r)
```

Represents a referent system for variable assignments.

The ReferentSystem class maintains associations between variables and integer pegs.

Definition at line 65 of file referent_system.cpp.

4.3.1.10 subsistsIn() [1/2]

```
bool iif_sadaf::talk::GSV::subsistsIn (
            const InformationState & s1,
            const InformationState & s2)
```

Checks if an information state subsists within another.

An information state s1 subsists in s2 if all possibilities in s1 have corresponding possibilities in s2.

Parameters

s1	The potential subsisting state.
s2	The state in which s1 may subsist.

Returns

True if s1 subsists in s2, false otherwise.

Definition at line 213 of file information_state.cpp.

4.3.1.11 subsistsIn() [2/2]

```
bool iif_sadaf::talk::GSV::subsistsIn (
            const Possibility & p,
            const InformationState & s)
```

Checks if a possibility subsists in an information state.

A possibility subsists in an information state if at least one of its descendants exists within the state.

Parameters

p	The possibility to check.
s	The information state.

Returns

True if p subsists in s, false otherwise.

Definition at line 197 of file information_state.cpp.

4.3.1.12 termDenotation()

```
int iif_sadaf::talk::GSV::termDenotation (
            std::string_view term,
            int w,
            const IModel & m)
```

Retrieves the denotation of a term in a given world within a model.

Parameters

term	The term to be interpreted.
w	The world in which the term is interpreted.
m	The model containing the interpretation.

Returns

The assigned individual for the term in the given world.

Exceptions

std::out_of_range	if the term does not exist.

Definition at line 14 of file semantic_relations.cpp.

4.3.1.13 update()

```
InformationState iif_sadaf::talk::GSV::update (
            const InformationState & input_state,
            std::string_view variable,
            int individual)
```

Updates the information state with a new variable-individual assignment.

Creates a new information state where each possibility has been updated with the given variable-individual assignment.

Parameters

input_state	The original information state.
variable	The variable to be added or updated.
individual	The individual assigned to the variable.

Returns

A new updated information state.

Definition at line 109 of file information_state.cpp.

4.3.1.14 variableDenotation()

```
int iif_sadaf::talk::GSV::variableDenotation (
            std::string_view variable,
            const Possibility & p)
```

Retrieves the denotation of a variable in a given possibility.

Parameters

variable	The variable to be interpreted.
p	The possibility containing the variable's assignment.

Returns

The assigned individual for the variable.

Exceptions

std::out_of_range	if the variable has no associated peg.

Definition at line 41 of file semantic_relations.cpp.

Chapter 5

Class Documentation

5.1 iif_sadaf::talk::GSV::Evaluator Struct Reference

Represents an evaluator for logical expressions.

`#include <evaluator.hpp>`

Public Member Functions

- InformationState operator() (std::shared_ptr< UnaryNode > expr, std::variant< InformationState > state) const

  Evaluates a unary logical expression on an InformationState.
- InformationState operator() (std::shared_ptr< BinaryNode > expr, std::variant< InformationState > state) const

  Evaluates a binary logical expression on an InformationState.
- InformationState operator() (std::shared_ptr< QuantificationNode > expr, std::variant< InformationState > state) const

  Evaluates a quantified expression on an InformationState.
- InformationState operator() (std::shared_ptr< IdentityNode > expr, std::variant< InformationState > state) const

  Evaluates an identity expression, filtering based on variable or term equality.
- InformationState operator() (std::shared_ptr< PredicationNode > expr, std::variant< InformationState > state) const

  Evaluates a predicate expression by filtering states based on predicate denotation.

5.1.1 Detailed Description

Represents an evaluator for logical expressions.

The Evaluator struct applies logical operations on `InformationState` objects using the visitor pattern. It evaluates different types of logical expressions, including unary, binary, quantification, identity, and predication nodes. The evaluation modifies or filters the given `InformationState` based on the logical rules applied.

Definition at line 19 of file evaluator.hpp.

5.1.2 Member Function Documentation

5.1.2.1 operator()() [1/5]

```
InformationState iif_sadaf::talk::GSV::Evaluator::operator() (
            std::shared_ptr< BinaryNode > expr,
            std::variant< InformationState > state) const
```

Evaluates a binary logical expression on an InformationState.

Processes logical operations such as conjunction, disjunction, and implication, modifying the state accordingly.

Parameters

expr	The binary expression to evaluate.
state	The current information state.

Returns

The modified InformationState after applying the operation.

Exceptions

std::invalid_argument	if the operator is invalid.

Definition at line 73 of file evaluator.cpp.

5.1.2.2 operator()() [2/5]

```
InformationState iif_sadaf::talk::GSV::Evaluator::operator() (
            std::shared_ptr< IdentityNode > expr,
            std::variant< InformationState > state) const
```

Evaluates an identity expression, filtering based on variable or term equality.

Compares the denotation of two terms or variables and retains only the possibilities where they are equal.

May throw std::out_of_range if either the LHS or the RHS of the identity lack an interpretation in the base model for the information state, or are variables without a binding quantifier or a proper anaphoric antecedent.

Parameters

expr	The identity expression to evaluate.
state	The current information state.

Returns

The filtered InformationState after applying identity conditions.

Definition at line 183 of file evaluator.cpp.

5.1.2.3 operator()() [3/5]

```
InformationState iif_sadaf::talk::GSV::Evaluator::operator() (
            std::shared_ptr< PredicationNode > expr,
            std::variant< InformationState > state) const
```

Evaluates a predicate expression by filtering states based on predicate denotation.

Checks if a given predicate holds in the current world and filters possibilities accordingly.

May throw std::out_of_range if (i) any argument to the predicate lacks an interpretation in the base model for the information state, or is a variable without a binding quantifier or a proper anaphoric antecedent, or (ii) the predicate lacks an interpretation in the base model for the information state.

Parameters

expr	The predicate expression to evaluate.
state	The current information state.

Returns

The filtered InformationState after evaluating the predicate.

Definition at line 213 of file evaluator.cpp.

5.1.2.4 operator()() [4/5]

```
InformationState iif_sadaf::talk::GSV::Evaluator::operator() (
            std::shared_ptr< QuantificationNode > expr,
            std::variant< InformationState > state) const
```

Evaluates a quantified expression on an InformationState.

Handles existential and universal quantifiers by iterating over possible individuals in the model and updating the state accordingly.

Parameters

expr	The quantification expression to evaluate.
state	The current information state.

Returns

The modified InformationState after applying the quantification.

Exceptions

std::invalid_argument	if the quantifier is invalid.

Definition at line 125 of file evaluator.cpp.

5.1.2.5 operator()() [5/5]

```
InformationState iif_sadaf::talk::GSV::Evaluator::operator() (
            std::shared_ptr< UnaryNode > expr,
            std::variant< InformationState > state) const
```

Evaluates a unary logical expression on an InformationState.

Applies an operator (such as necessity, possibility, or negation) to modify the given state accordingly.

Parameters

expr	The unary expression to evaluate.
state	The current information state.

Returns

The modified InformationState after applying the operation.

Exceptions

std::invalid_argument	if the operator is invalid.

Definition at line 37 of file evaluator.cpp.

The documentation for this struct was generated from the following files:

- evaluator.hpp
- evaluator.cpp

5.2 iif_sadaf::talk::GSV::IModel Struct Reference

Interface for class representing a model for QML without accessiblity.

`#include <imodel.hpp>`

Public Member Functions

- virtual int world_cardinality () const =0
- virtual int domain_cardinality () const =0
- virtual int termInterpretation (std::string_view term, int world) const =0
- virtual const std::set< std::vector< int > > & predicateInterpretation (std::string_view predicate, int world) const =0
- virtual ~IModel ()

5.2.1 Detailed Description

Interface for class representing a model for QML without accessiblity.

The IModel interface defines the minimal requirements on any implementation of a QML model that works with the GSV evaluator library.

Any such implementation should contain four functions:

- a function retrieving the cardinality of the set W of worlds
- a function retrieving the cardinality of the domain of individuals
- a function that retrieves, for any possible world in W, the interpretation of a singular term at that world (represented by an `int`)
- a function that retrieves, for any possible world in W, the interpretation of a predicate at that world (represented by a `std::set<std::vector<int>>`)

Definition at line 21 of file imodel.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ~IModel()

```
virtual iif_sadaf::talk::GSV::IModel::~IModel ()  [inline], [virtual]
```

Definition at line 27 of file imodel.hpp.

5.2.3 Member Function Documentation

5.2.3.1 domain_cardinality()

```
virtual int iif_sadaf::talk::GSV::IModel::domain_cardinality () const  [pure virtual]
```

5.2.3.2 predicateInterpretation()

```
virtual const std::set< std::vector< int > > & iif_sadaf::talk::GSV::IModel::predicate↩
Interpretation (
             std::string_view predicate,
             int world) const  [pure virtual]
```

5.2.3.3 termInterpretation()

```
virtual int iif_sadaf::talk::GSV::IModel::termInterpretation (
             std::string_view term,
             int world) const  [pure virtual]
```

5.2.3.4 world_cardinality()

```
virtual int iif_sadaf::talk::GSV::IModel::world_cardinality () const  [pure virtual]
```

The documentation for this struct was generated from the following file:

- imodel.hpp

5.3 iif_sadaf::talk::GSV::InformationState Struct Reference

Represents an information state based on a given model.

```
#include <information_state.hpp>
```

Public Member Functions

- InformationState (const IModel &model, bool create_possibilities=true)

  Constructs an InformationState based on a given model.
- bool empty () const

  Checks if the information state is empty.
- void clear ()

  Clears all possibilities from the information state.
- std::set< Possibility >::iterator begin ()

  Returns an iterator to the beginning of the possibilities set.
- std::set< Possibility >::iterator end ()

  Returns an iterator to the end of the possibilities set.
- std::set< Possibility >::iterator erase (std::set< Possibility >::iterator it)

  Removes a possibility from the set.
- bool contains (const Possibility &p) const

  Checks if a given possibility is present in the information state.

Public Attributes

- std::set< Possibility > possibilities = {}
- const IModel & model

5.3.1 Detailed Description

Represents an information state based on a given model.

The InformationState class maintains a set of possibilities and provides operations to manage them.

Definition at line 18 of file information_state.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 InformationState()

```
iif_sadaf::talk::GSV::InformationState::InformationState (
            const IModel & model,
            bool create_possibilities = true)
```

Constructs an InformationState based on a given model.

Initializes the information state and optionally populates it with possibilities.

Parameters

model	The model to which this information state belongs.
create_possibilities	Whether to initialize the possibilities set.

Definition at line 16 of file information_state.cpp.

5.3.3 Member Function Documentation

5.3.3.1 begin()

```
std::set< Possibility >::iterator iif_sadaf::talk::GSV::InformationState::begin ()
```

Returns an iterator to the beginning of the possibilities set.

Returns

Iterator to the beginning of the possibilities.

Definition at line 55 of file information_state.cpp.

5.3.3.2 clear()

```
void iif_sadaf::talk::GSV::InformationState::clear ()
```

Clears all possibilities from the information state.

Definition at line 45 of file information_state.cpp.

5.3.3.3 contains()

```
bool iif_sadaf::talk::GSV::InformationState::contains (
            const Possibility & p) const
```

Checks if a given possibility is present in the information state.

Parameters

p	The possibility to check.

Returns

True if the possibility is present, false otherwise.

Definition at line 88 of file information_state.cpp.

5.3.3.4 empty()

```
bool iif_sadaf::talk::GSV::InformationState::empty () const
```

Checks if the information state is empty.

Returns

True if there are no possibilities, false otherwise.

Definition at line 37 of file information_state.cpp.

5.3.3.5 end()

```
std::set< Possibility >::iterator iif_sadaf::talk::GSV::InformationState::end ()
```

Returns an iterator to the end of the possibilities set.

Returns

Iterator to the end of the possibilities.

Definition at line 65 of file information_state.cpp.

5.3.3.6 erase()

```
std::set< Possibility >::iterator iif_sadaf::talk::GSV::InformationState::erase (
            std::set< Possibility >::iterator it)
```

Removes a possibility from the set.

Parameters

it	Iterator pointing to the possibility to erase.

Returns

Iterator following the last removed element.

Definition at line 76 of file information_state.cpp.

5.3.4 Member Data Documentation

5.3.4.1 model

```
const IModel& iif_sadaf::talk::GSV::InformationState::model
```

Definition at line 31 of file information_state.hpp.

5.3.4.2 possibilities

```
std::set<Possibility> iif_sadaf::talk::GSV::InformationState::possibilities = {}
```

Definition at line 30 of file information_state.hpp.

The documentation for this struct was generated from the following files:

- information_state.hpp
- information_state.cpp

5.4 iif_sadaf::talk::GSV::Possibility Struct Reference

Represents a possibility as understood in the underlying semantics.

`#include <possibility.hpp>`

Public Member Functions

- Possibility (std::shared_ptr< ReferentSystem > r_system, int world)
  Constructs a Possibility with a given referent system and world index.
- int getAssignment (int peg) const
  Retrieves the individual assigned to a given peg.
- void update (std::string_view variable, int individual)
  Updates the assignment of a variable to an individual.

Public Attributes

- std::shared_ptr< ReferentSystem > referentSystem
- std::unordered_map< int, int > assignment
- int world

5.4.1 Detailed Description

Represents a possibility as understood in the underlying semantics.

Possibilities are just tuples of a referent system, an assignment of individuals to pegs, and a possible world.

The class also contains a few convenience functions for handling the first two components.

Definition at line 18 of file possibility.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Possibility()

```
iif_sadaf::talk::GSV::Possibility::Possibility (
            std::shared_ptr< ReferentSystem > r_system,
            int world)
```

Constructs a Possibility with a given referent system and world index.

Parameters

r_system	Shared pointer to the referent system.
world	The index of the possible world.

Definition at line 13 of file possibility.cpp.

5.4.3 Member Function Documentation

5.4.3.1 getAssignment()

```
int iif_sadaf::talk::GSV::Possibility::getAssignment (
            int peg) const
```

Retrieves the individual assigned to a given peg.

Parameters

peg	The peg whose assigned individual is to be retrieved.

Returns

The assigned individual, or -1 if the peg is not assigned.

Definition at line 40 of file possibility.cpp.

5.4.3.2 update()

```
void iif_sadaf::talk::GSV::Possibility::update (
            std::string_view variable,
            int individual)
```

Updates the assignment of a variable to an individual.

The variable is first updated in the associated referent system. Then, the assignment is modified to map the peg of the variable to the new individual.

Parameters

variable	The variable to update.
individual	The new individual assigned to the variable.

Definition at line 28 of file possibility.cpp.

5.4.4 Member Data Documentation

5.4.4.1 assignment

```
std::unordered_map<int, int> iif_sadaf::talk::GSV::Possibility::assignment
```

Definition at line 26 of file possibility.hpp.

5.4.4.2 referentSystem

```
std::shared_ptr<ReferentSystem> iif_sadaf::talk::GSV::Possibility::referentSystem
```

Definition at line 25 of file possibility.hpp.

5.4.4.3 world

```
int iif_sadaf::talk::GSV::Possibility::world
```

Definition at line 27 of file possibility.hpp.

The documentation for this struct was generated from the following files:

- possibility.hpp
- possibility.cpp

5.5 iif_sadaf::talk::GSV::ReferentSystem Struct Reference

Represents a referent system for variable assignments.

`#include <referent_system.hpp>`

Public Member Functions

- int range () const

  Returns the range (number of pegs) of the ReferentSystem.
- std::set< std::string_view > domain () const

  Returns the domain (set of variables) of the ReferentSystem.
- int value (std::string_view variable) const

  Retrieves the peg value associated with a given variable.
- void update (std::string_view variable)

Public Attributes

- int pegs = 0
- std::unordered_map< std::string_view, int > variablePegAssociation = {}

5.5.1 Detailed Description

Represents a referent system for variable assignments.

The ReferentSystem class maintains associations between variables and integer pegs.

Definition at line 15 of file referent_system.hpp.

5.5.2 Member Function Documentation

5.5.2.1 domain()

```
std::set< std::string_view > iif_sadaf::talk::GSV::ReferentSystem::domain () const
```

Returns the domain (set of variables) of the ReferentSystem.

Returns

A set of string views representing the variables in the referent system.

Definition at line 28 of file referent_system.cpp.

5.5.2.2 range()

```
int iif_sadaf::talk::GSV::ReferentSystem::range () const
```

Returns the range (number of pegs) of the ReferentSystem.

Since the range of a referent system is always an initial segment of the natural numbers, the number of pegs represents the range of the referent system.

Returns

The number of pegs in the referent system.

Definition at line 17 of file referent_system.cpp.

5.5.2.3 update()

```
void iif_sadaf::talk::GSV::ReferentSystem::update (
            std::string_view variable)
```

Definition at line 55 of file referent_system.cpp.

5.5.2.4 value()

```
int iif_sadaf::talk::GSV::ReferentSystem::value (
            std::string_view variable) const
```

Retrieves the peg value associated with a given variable.

Parameters

variable	The variable whose peg value is to be retrieved.

Returns

The peg value associated with the variable.

Exceptions

std::out_of_range	If the variable has no associated peg.

Definition at line 45 of file referent_system.cpp.

5.5.3 Member Data Documentation

5.5.3.1 pegs

```
int iif_sadaf::talk::GSV::ReferentSystem::pegs = 0
```

Definition at line 22 of file referent_system.hpp.

5.5.3.2 variablePegAssociation

```
std::unordered_map<std::string_view, int> iif_sadaf::talk::GSV::ReferentSystem::variablePeg↩
Association = {}
```

Definition at line 23 of file referent_system.hpp.

The documentation for this struct was generated from the following files:

- referent_system.hpp
- referent_system.cpp

Chapter 6

File Documentation

6.1 evaluator.hpp File Reference

```
#include <functional>
#include "expression.hpp"
#include "information_state.hpp"
```

Classes

- struct iif_sadaf::talk::GSV::Evaluator

  Represents an evaluator for logical expressions.

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

6.2 evaluator.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <functional>
00004
00005 #include "expression.hpp"
00006 #include "information_state.hpp"
00007
00008 namespace iif_sadaf::talk::GSV {
00009
00019 struct Evaluator {
00020     InformationState operator()(std::shared_ptr<UnaryNode> expr, std::variant<InformationState> state)
      const;
00021     InformationState operator()(std::shared_ptr<BinaryNode> expr, std::variant<InformationState>
      state) const;
00022     InformationState operator()(std::shared_ptr<QuantificationNode> expr,
      std::variant<InformationState> state) const;
00023     InformationState operator()(std::shared_ptr<IdentityNode> expr, std::variant<InformationState>
      state) const;
00024     InformationState operator()(std::shared_ptr<PredicationNode> expr, std::variant<InformationState>
      state) const;
00025 };
00026
00027 }
```

6.3 imodel.hpp File Reference

```
#include <set>
#include <string_view>
#include <vector>
```

Classes

- struct iif_sadaf::talk::GSV::IModel

  Interface for class representing a model for QML without accessiblity.

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

6.4 imodel.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <set>
00004 #include <string_view>
00005 #include <vector>
00006
00007 namespace iif_sadaf::talk::GSV {
00008
00021 struct IModel {
00022 public:
00023     virtual int world_cardinality() const = 0;
00024     virtual int domain_cardinality() const = 0;
00025     virtual int termInterpretation(std::string_view term, int world) const = 0;
00026     virtual const std::set<std::vector<int>& predicateInterpretation(std::string_view predicate, int
    world) const = 0;
00027     virtual ~IModel() {};
00028 };
00029
00030 }
```

6.5 information_state.hpp File Reference

```
#include <set>
#include <string>
#include <string_view>
#include "model.hpp"
#include "possibility.hpp"
```

Classes

- struct iif_sadaf::talk::GSV::InformationState

  Represents an information state based on a given model.

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- InformationState iif_sadaf::talk::GSV::update (const InformationState &input_state, std::string_view variable, int individual)

  Updates the information state with a new variable-individual assignment.
- bool iif_sadaf::talk::GSV::extends (const InformationState &s2, const InformationState &s1)

  Determines if one information state extends another.
- std::string iif_sadaf::talk::GSV::str (const InformationState &state)
- bool iif_sadaf::talk::GSV::isDescendantOf (const Possibility &p2, const Possibility &p1, const InformationState &s)

  Determines if one possibility is a descendant of another within an information state.
- bool iif_sadaf::talk::GSV::subsistsIn (const Possibility &p, const InformationState &s)

  Checks if a possibility subsists in an information state.
- bool iif_sadaf::talk::GSV::subsistsIn (const InformationState &s1, const InformationState &s2)

  Checks if an information state subsists within another.

6.6 information_state.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <set>
00004 #include <string>
00005 #include <string_view>
00006
00007 #include "model.hpp"
00008 #include "possibility.hpp"
00009
00010 namespace iif_sadaf::talk::GSV {
00011
00018 struct InformationState {
00019 public:
00020     InformationState(const IModel& model, bool create_possibilities = true);
00021
00022     bool empty() const;
00023     void clear();
00024
00025     std::set<Possibility>::iterator begin();
00026     std::set<Possibility>::iterator end();
00027     std::set<Possibility>::iterator erase(std::set<Possibility>::iterator it);
00028     bool contains(const Possibility& p) const;
00029
00030     std::set<Possibility> possibilities = {};
00031     const IModel& model;
00032 };
00033
00034 InformationState update(const InformationState& input_state, std::string_view variable, int
      individual);
00035 bool extends(const InformationState& s2, const InformationState& s1);
00036
00037 std::string str(const InformationState& state);
00038
00039 bool isDescendantOf(const Possibility& p2, const Possibility& p1, const InformationState& s);
00040 bool subsistsIn(const Possibility& p, const InformationState& s);
00041 bool subsistsIn(const InformationState& s1, const InformationState& s2);
00042
00043 }
```

6.7 possibility.hpp File Reference

```
#include <memory>
#include <string>
#include <unordered_map>
#include "referent_system.hpp"
```

Classes

- struct iif_sadaf::talk::GSV::Possibility

  Represents a possibility as understood in the underlying semantics.

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- bool iif_sadaf::talk::GSV::extends (const Possibility &p2, const Possibility &p1)

  Determines whether one Possibility extends another.
- bool iif_sadaf::talk::GSV::operator< (const Possibility &p1, const Possibility &p2)
- std::string iif_sadaf::talk::GSV::str (const Possibility &p)

6.8 possibility.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <memory>
00004 #include <string>
00005 #include <unordered_map>
00006
00007 #include "referent_system.hpp"
00008
00009 namespace iif_sadaf::talk::GSV {
00010
00018 struct Possibility {
00019 public:
00020     Possibility(std::shared_ptr<ReferentSystem> r_system, int world);
00021
00022     int getAssignment(int peg) const;
00023     void update(std::string_view variable, int individual);
00024
00025     std::shared_ptr<ReferentSystem> referentSystem;
00026     std::unordered_map<int, int> assignment;
00027     int world;
00028 };
00029
00030 bool extends(const Possibility& p2, const Possibility& p1);
00031 bool operator<(const Possibility& p1, const Possibility& p2);
00032 std::string str(const Possibility& p);
00033
00034 }
```

6.9 referent_system.hpp File Reference

```
#include <set>
#include <string>
#include <string_view>
#include <unordered_map>
```

Classes

- struct iif_sadaf::talk::GSV::ReferentSystem

  Represents a referent system for variable assignments.

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- bool iif_sadaf::talk::GSV::extends (const ReferentSystem &r2, const ReferentSystem &r1)

  Determines whether one ReferentSystem extends another.
- std::string iif_sadaf::talk::GSV::str (const ReferentSystem &r)

  Represents a referent system for variable assignments.

6.10 referent_system.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <set>
00004 #include <string>
00005 #include <string_view>
00006 #include <unordered_map>
00007
00008 namespace iif_sadaf::talk::GSV {
00009
00015 struct ReferentSystem {
00016 public:
00017     int range() const;
00018     std::set<std::string_view> domain() const;
00019     int value(std::string_view variable) const;
00020     void update(std::string_view variable);
00021
00022     int pegs = 0;
00023     std::unordered_map<std::string_view, int> variablePegAssociation = {};
00024 };
00025
00026 bool extends(const ReferentSystem& r2, const ReferentSystem& r1);
00027 std::string str(const ReferentSystem& r);
00028
00029 }
```

6.11 semantic_relations.hpp File Reference

```
#include <string>
#include "information_state.hpp"
#include "model.hpp"
#include "possibility.hpp"
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- int iif_sadaf::talk::GSV::termDenotation (std::string_view term, int w, const IModel &m)

  Retrieves the denotation of a term in a given world within a model.
- const std::set< std::vector< int > > & iif_sadaf::talk::GSV::predicateDenotation (std::string_view predicate, int w, const IModel &m)

  Retrieves the denotation of a predicate in a given world within a model.
- int iif_sadaf::talk::GSV::variableDenotation (std::string_view variable, const Possibility &p)

  Retrieves the denotation of a variable in a given possibility.

6.12 semantic_relations.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <string>
00004
00005 #include "information_state.hpp"
00006 #include "model.hpp"
00007 #include "possibility.hpp"
00008
00009 namespace iif_sadaf::talk::GSV {
00010
00011 int termDenotation(std::string_view term, int world, const IModel& model);
00012 const std::set<std::vector<int>& predicateDenotation(std::string_view predicate, int world, const
      IModel& model);
00013 int variableDenotation(std::string_view variable, const Possibility& possibility);
00014
00015 }
```

6.13 evaluator.cpp File Reference

```
#include "evaluator.hpp"
#include <algorithm>
#include <functional>
#include <stdexcept>
#include <ranges>
#include "semantic_relations.hpp"
#include "variable.hpp"
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

6.14 evaluator.cpp

Go to the documentation of this file.

```
00001 #include "evaluator.hpp"
00002
00003 #include <algorithm>
00004 #include <functional>
00005 #include <stdexcept>
00006 #include <ranges>
00007
00008 #include "semantic_relations.hpp"
00009 #include "variable.hpp"
00010
00011 namespace iif_sadaf::talk::GSV {
00012
00013 namespace {
00014     void filter(InformationState& state, const std::function<bool(const Possibility&)>& predicate) {
00015         for (auto it = state.begin(); it != state.end(); ) {
00016             if (!predicate(*it)) {
00017                 it = state.erase(it);
00018             }
00019             else {
00020                 ++it;
00021             }
00022         }
00023     }
00024 }
00025
00037 InformationState Evaluator::operator()(std::shared_ptr<UnaryNode> expr, std::variant<InformationState>
      state) const
00038 {
00039     InformationState hypothetical = std::visit(Evaluator(), expr->scope, state);
00040     InformationState& s = std::get<InformationState>(state);
00041
00042     if (expr->op == Operator::E_POS) {
00043         if (hypothetical.empty()) {
00044             s.clear();
00045         }
00046     }
00047     else if (expr->op == Operator::E_NEC) {
00048         if (!subsistsIn(s, hypothetical)) {
00049             s.clear();
00050         }
00051     }
00052     else if (expr->op == Operator::NEG) {
00053         filter(s, [&](const Possibility& p) -> bool { return !subsistsIn(p, hypothetical); });
00054     }
00055     else {
00056         throw(std::invalid_argument("Invalid operator for unary formula"));
00057     }
00058
00059     return s;
00060 }
00061
00073 InformationState Evaluator::operator()(std::shared_ptr<BinaryNode> expr,
      std::variant<InformationState> state) const
00074 {
00075     if (expr->op == Operator::CON) {
00076         return std::visit(Evaluator(), expr->rhs,
      std::variant<InformationState>(std::visit(Evaluator(), expr->lhs, state)));
00077     }
00078
00079     InformationState& s = std::get<InformationState>(state);
00080     InformationState hypothetical_lhs = std::visit(Evaluator(), expr->lhs, state);
00081
00082     if (expr->op == Operator::DIS) {
00083         InformationState hypothetical_rhs = std::visit(Evaluator(), expr->rhs,
      std::variant<InformationState>(std::visit(Evaluator(), negate(expr->lhs), state)));
00084
00085         const auto in_lhs_or_in_rhs = [&](const Possibility& p) -> bool {
00086             return hypothetical_lhs.contains(p) || hypothetical_rhs.contains(p);
00087         };
```

```
00088
00089         filter(s, in_lhs_or_in_rhs);
00090     }
00091     else if (expr->op == Operator::IMP) {
00092         InformationState hypothetical_consequent = std::visit(Evaluator(), expr->rhs,
      std::variant<InformationState>(hypothetical_lhs));
00093
00094         auto all_descendants_subsist = [&](const Possibility& p) -> bool {
00095             auto not_descendant_or_subsists = [&](const Possibility& p_star) -> bool {
00096                 return !isDescendantOf(p_star, p, hypothetical_lhs) || subsistsIn(p_star,
      hypothetical_consequent);
00097             };
00098             return std::ranges::all_of(hypothetical_lhs.possibilities, not_descendant_or_subsists);
00099         };
00100
00101         const auto if_subsists_all_descendants_do = [&](const Possibility& p) -> bool {
00102             return !subsistsIn(p, hypothetical_lhs) || all_descendants_subsist(p);
00103         };
00104
00105         filter(s, if_subsists_all_descendants_do);
00106     }
00107     else {
00108         throw(std::invalid_argument("Invalid operator for binary formula"));
00109     }
00110
00111     return s;
00112 }
00113
00125 InformationState Evaluator::operator()(std::shared_ptr<QuantificationNode> expr,
      std::variant<InformationState> state) const
00126 {
00127     InformationState& s = std::get<InformationState>(state);
00128
00129     if (expr->quantifier == Quantifier::EXISTENTIAL) {
00130         std::vector<InformationState> all_state_variants;
00131
00132         for (int i : std::views::iota(0, s.model.domain_cardinality())) {
00133             InformationState s_variant = update(s, expr->variable, i);
00134             all_state_variants.push_back(std::visit(Evaluator(), expr->scope,
      std::variant<InformationState>(s_variant)));
00135         }
00136
00137         InformationState output(s.model, false);
00138         for (const auto& state_variant : all_state_variants) {
00139             for (const auto& p : state_variant.possibilities) {
00140                 output.possibilities.insert(p);
00141             }
00142         }
00143
00144         return output;
00145     }
00146     else if (expr->quantifier == Quantifier::UNIVERSAL) {
00147         std::vector<InformationState> all_hypothetical_updates;
00148
00149         for (int d : std::views::iota(0, s.model.domain_cardinality())) {
00150             InformationState hypothetical = std::visit(Evaluator(), expr->scope,
      std::variant<InformationState>(update(s, expr->variable, d)));
00151             all_hypothetical_updates.push_back(hypothetical);
00152         }
00153
00154         const auto subsists_in_all_hyp_updates = [&](const Possibility& p) -> bool {
00155             const auto p_subsists_in_hyp_update = [&](const InformationState& hypothetical) -> bool {
00156                 return subsistsIn(p, hypothetical);
00157             };
00158             return std::ranges::all_of(all_hypothetical_updates, p_subsists_in_hyp_update);
00159         };
00160
00161         filter(s, subsists_in_all_hyp_updates);
00162     }
00163     else {
00164         throw(std::invalid_argument("Invalid quantifier"));
00165     }
00166     return s;
00167 }
00168
00183 InformationState Evaluator::operator()(std::shared_ptr<IdentityNode> expr,
      std::variant<InformationState> state) const
00184 {
00185     InformationState& s = std::get<InformationState>(state);
00186
00187     auto assigns_same_denotation = [&](const Possibility& p) -> bool {
00188         const int lhs_denotation = isVariable(expr->lhs) ? variableDenotation(expr->lhs, p) :
      termDenotation(expr->lhs, p.world, s.model);
00189         const int rhs_denotation = isVariable(expr->lhs) ? variableDenotation(expr->rhs, p) :
      termDenotation(expr->rhs, p.world, s.model);
00190         return lhs_denotation == rhs_denotation;
00191     };
```

```
00192
00193     filter(s, assigns_same_denotation);
00194
00195     return s;
00196 }
00197
00213 InformationState Evaluator::operator()(std::shared_ptr<PredicationNode> expr,
      std::variant<InformationState> state) const
00214 {
00215     InformationState& s = std::get<InformationState>(state);
00216
00217     auto tuple_in_extension = [&](const Possibility& p) -> bool {
00218         std::vector<int> tuple;
00219
00220         for (const std::string& argument : expr->arguments) {
00221             const int denotation = isVariable(argument) ? variableDenotation(argument, p) :
      termDenotation(argument, p.world, s.model);
00222             tuple.push_back(denotation);
00223         }
00224
00225         return predicateDenotation(expr->predicate, p.world, s.model).contains(tuple);
00226     };
00227
00228     filter(s, tuple_in_extension);
00229
00230     return s;
00231 }
00232
00233 }
```

6.15 information_state.cpp File Reference

```
#include "information_state.hpp"
#include <algorithm>
#include <memory>
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- InformationState iif_sadaf::talk::GSV::update (const InformationState &input_state, std::string_view variable, int individual)

  Updates the information state with a new variable-individual assignment.
- bool iif_sadaf::talk::GSV::extends (const InformationState &s2, const InformationState &s1)

  Determines if one information state extends another.
- std::string iif_sadaf::talk::GSV::str (const InformationState &state)
- bool iif_sadaf::talk::GSV::isDescendantOf (const Possibility &p2, const Possibility &p1, const InformationState &s)

  Determines if one possibility is a descendant of another within an information state.
- bool iif_sadaf::talk::GSV::subsistsIn (const Possibility &p, const InformationState &s)

  Checks if a possibility subsists in an information state.
- bool iif_sadaf::talk::GSV::subsistsIn (const InformationState &s1, const InformationState &s2)

  Checks if an information state subsists within another.

6.16 information_state.cpp

Go to the documentation of this file.

```
00001 #include "information_state.hpp"
00002
00003 #include <algorithm>
00004 #include <memory>
00005
00006 namespace iif_sadaf::talk::GSV {
00007
00016 InformationState::InformationState(const IModel& model, bool create_possibilities)
00017     : possibilities()
00018     , model(model)
00019 {
00020     auto r_system = std::make_shared<ReferentSystem>();
00021
00022     if (!create_possibilities) {
00023         return;
00024     }
00025
00026     const int number_of_worlds = model.domain_cardinality();
00027     for (int i = 0; i < number_of_worlds; ++i) {
00028         possibilities.insert(Possibility(r_system, i));
00029     }
00030 }
00031
00037 bool InformationState::empty() const
00038 {
00039     return possibilities.empty();
00040 }
00041
00045 void InformationState::clear()
00046 {
00047     possibilities.clear();
00048 }
00049
00055 std::set<Possibility>::iterator InformationState::begin()
00056 {
00057     return possibilities.begin();
00058 }
00059
00065 std::set<Possibility>::iterator InformationState::end()
00066 {
00067     return possibilities.end();
00068 }
00069
00076 std::set<Possibility>::iterator InformationState::erase(std::set<Possibility>::iterator it)
00077 {
00078     return possibilities.erase(it);
00079 }
00080
00081
00088 bool InformationState::contains(const Possibility& p) const
00089 {
00090     return possibilities.contains(p);
00091 }
00092
00093 /*
00094 * NON-MEMBER INTERFACE FUNCTIONS
00095 */
00096
00097
00109 InformationState update(const InformationState& input_state, std::string_view variable, int
      individual)
00110 {
00111     InformationState output_state(input_state.model, false);
00112
00113     auto r_star = std::make_shared<ReferentSystem>();
00114
00115     for (const auto& p : input_state.possibilities) {
00116         Possibility p_star(r_star, p.world);
00117         p_star.assignment = p.assignment;
00118         r_star->pegs = p.referentSystem->pegs;
00119         for (const auto& map : p.referentSystem->variablePegAssociation) {
00120             auto var = map.first;
00121             int peg = map.second;
00122             r_star->variablePegAssociation[var] = peg;
00123         }
00124
00125         p_star.update(variable, individual);
00126
00127         output_state.possibilities.insert(p_star);
00128     }
00129
00130     return output_state;
```

```
00131 }
00132
00142 bool extends(const InformationState& s2, const InformationState& s1)
00143 {
00144     const auto extends_possibility_in_s1 = [&](const Possibility& p2) -> bool {
00145         const auto is_extended_by_p2 = [&](const Possibility& p1) -> bool {
00146             return extends(p2, p1);
00147         };
00148         return std::ranges::any_of(s1.possibilities, is_extended_by_p2);
00149     };
00150
00151     return std::ranges::all_of(s2.possibilities, extends_possibility_in_s1);
00152 }
00153
00154 /*
00155 * NON-INTERFACE FUNCTIONS
00156 */
00157
00158 std::string str(const InformationState& state)
00159 {
00160     std::string desc;
00161
00162     desc += "--------------------\n";
00163     for (const Possibility& p : state.possibilities) {
00164         desc += str(p);
00165         desc += "--------------------\n";
00166     }
00167
00168     desc.pop_back();
00169
00170     return desc;
00171 }
00172
00183 bool isDescendantOf(const Possibility& p2, const Possibility& p1, const InformationState& s)
00184 {
00185     return s.possibilities.contains(p2) && (extends(p2, p1));
00186 }
00187
00197 bool subsistsIn(const Possibility& p, const InformationState& s)
00198 {
00199     const auto is_descendant_of_p_in_s = [&](const Possibility& p1) -> bool { return
      isDescendantOf(p1, p, s); };
00200     return std::ranges::any_of(s.possibilities, is_descendant_of_p_in_s);
00201 }
00202
00203
00213 bool subsistsIn(const InformationState& s1, const InformationState& s2)
00214 {
00215     const auto subsists_in_s2 = [&](const Possibility& p) -> bool { return subsistsIn(p, s2); };
00216     return std::ranges::all_of(s1.possibilities, subsists_in_s2);
00217 }
00218
00219 }
```

6.17 possibility.cpp File Reference

```
#include "possibility.hpp"
#include <algorithm>
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- bool iif_sadaf::talk::GSV::extends (const Possibility &p2, const Possibility &p1)

  Determines whether one Possibility extends another.
- bool iif_sadaf::talk::GSV::operator< (const Possibility &p1, const Possibility &p2)
- std::string iif_sadaf::talk::GSV::str (const Possibility &p)

6.18 possibility.cpp

Go to the documentation of this file.

```
00001 #include "possibility.hpp"
00002
00003 #include <algorithm>
00004
00005 namespace iif_sadaf::talk::GSV {
00006
00013 Possibility::Possibility(std::shared_ptr<ReferentSystem> r_system, int world)
00014     : referentSystem(r_system)
00015     , assignment({})
00016     , world(world)
00017 { }
00018
00028 void Possibility::update(std::string_view variable, int individual)
00029 {
00030     referentSystem->update(variable);
00031     assignment[referentSystem->variablePegAssociation.at(variable)] = individual;
00032 }
00033
00040 int Possibility::getAssignment(int peg) const
00041 {
00042     if (!assignment.contains(peg)) {
00043         return -1;
00044     }
00045
00046     return assignment.at(peg);
00047 }
00048
00049 /*
00050 * NON-MEMBER FUNCTIONS
00051 */
00052
00064 bool extends(const Possibility& p2, const Possibility& p1)
00065 {
00066     const auto peg_is_new_or_maintains_assignment = [&](const std::pair<int, int>& map) -> bool {
00067         int peg = map.first;
00068         int individual = map.second;
00069
00070         return !p1.assignment.contains(peg) || (p1.getAssignment(peg) == p2.getAssignment(peg));
00071     };
00072
00073     return (p1.world == p2.world) && std::ranges::all_of(p2.assignment,
      peg_is_new_or_maintains_assignment);
00074 }
00075
00076 bool operator<(const Possibility& p1, const Possibility& p2)
00077 {
00078     return p1.world < p2.world;
00079 }
00080
00081 std::string str(const Possibility& p)
00082 {
00083     std::string desc = "[ ] Referent System:\n" + str(*p.referentSystem);
00084     desc += "[ ] Assignment function: \n";
00085
00086     if (p.assignment.empty()) {
00087         desc += "  [ empty ]\n";
00088
00089     }
00090     else {
00091         for (const auto& [peg, individual] : p.assignment) {
00092             desc += "  - peg_" + std::to_string(peg) + " -> e_" + std::to_string(individual) + "\n";
00093         }
00094     }
00095
00096     desc += "[ ] Possible world: w_" + std::to_string(p.world) + "\n";
00097
00098     return desc;
00099 }
00100
00101 }
```

6.19 referent_system.cpp File Reference

```
#include "referent_system.hpp"
#include <algorithm>
#include <stdexcept>
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- std::string iif_sadaf::talk::GSV::str (const ReferentSystem &r)

  Represents a referent system for variable assignments.
- bool iif_sadaf::talk::GSV::extends (const ReferentSystem &r2, const ReferentSystem &r1)

  Determines whether one ReferentSystem extends another.

6.20 referent_system.cpp

Go to the documentation of this file.

```
00001 #include "referent_system.hpp"
00002
00003 #include <algorithm>
00004 #include <stdexcept>
00005
00006 namespace iif_sadaf::talk::GSV {
00007
00017 int ReferentSystem::range() const
00018 {
00019     return pegs;
00020 }
00021
00028 std::set<std::string_view> ReferentSystem::domain() const
00029 {
00030     std::set<std::string_view> domain;
00031     for (const auto& [variable, peg] : variablePegAssociation) {
00032         domain.insert(variable);
00033     }
00034
00035     return domain;
00036 }
00037
00045 int ReferentSystem::value(std::string_view variable) const
00046 {
00047     if (!variablePegAssociation.contains(variable)) {
00048         std::string error_msg = "Variable " + std::string(variable) + " has no anaphoric antecedent of
      binding quantifier";
00049         throw(std::out_of_range(error_msg));
00050     }
00051
00052     return variablePegAssociation.at(variable);
00053 }
00054
00055 void ReferentSystem::update(std::string_view variable)
00056 {
00057     variablePegAssociation[variable] = ++pegs;
00058 }
00059
00065 std::string str(const ReferentSystem& r)
00066 {
00067     std::string desc = "Number of pegs: " + std::to_string(r.pegs) + "\n";
00068     desc += "Variable to peg association:\n";
00069
00070     if (r.variablePegAssociation.empty()) {
00071         desc += "  [ empty ]\n";
00072         return desc;
00073     }
00074
00075     for (const auto& [variable, peg] : r.variablePegAssociation) {
00076         desc += "  - " + std::string(variable) + " -> peg_" + std::to_string(peg) + "\n";
00077     }
00078
00079     return desc;
00080 }
00081
00096 bool extends(const ReferentSystem& r2, const ReferentSystem& r1)
```

```
00097 {
00098     if (r1.range() > r2.range()) {
00099         return false;
00100     }
00101
00102     std::set<std::string_view> domain_r1 = r1.domain();
00103     std::set<std::string_view> domain_r2 = r2.domain();
00104
00105     if (!std::ranges::includes(domain_r2, domain_r1)) {
00106         return false;
00107     }
00108
00109     const auto old_var_same_or_new_peg = [&](std::string_view variable) -> bool {
00110         return r1.value(variable) == r2.value(variable) || r1.range() <= r2.value(variable);
00111     };
00112
00113     if (!std::ranges::all_of(domain_r1, old_var_same_or_new_peg)) {
00114         return false;
00115     }
00116
00117     const auto new_var_new_peg = [&](std::string_view variable) -> bool {
00118         return domain_r1.contains(variable) || r1.range() <= r2.value(variable);
00119     };
00120
00121     return std::ranges::all_of(domain_r2, new_var_new_peg);
00122 }
00123
00124 }
```

6.21 semantic_relations.cpp File Reference

```
#include "semantic_relations.hpp"
```

Namespaces

- namespace iif_sadaf
- namespace iif_sadaf::talk
- namespace iif_sadaf::talk::GSV

Functions

- int iif_sadaf::talk::GSV::termDenotation (std::string_view term, int w, const IModel &m)

  Retrieves the denotation of a term in a given world within a model.
- const std::set< std::vector< int > > & iif_sadaf::talk::GSV::predicateDenotation (std::string_view predicate, int w, const IModel &m)

  Retrieves the denotation of a predicate in a given world within a model.
- int iif_sadaf::talk::GSV::variableDenotation (std::string_view variable, const Possibility &p)

  Retrieves the denotation of a variable in a given possibility.

6.22 semantic_relations.cpp

Go to the documentation of this file.

```
00001 #include "semantic_relations.hpp"
00002
00003 namespace iif_sadaf::talk::GSV {
00004
00014 int termDenotation(std::string_view term, int w, const IModel& m)
00015 {
00016     return m.termInterpretation(term, w);
00017 }
```

```
00018
00028 const std::set<std::vector<int>& predicateDenotation(std::string_view predicate, int w, const IModel&
      m)
00029 {
00030     return m.predicateInterpretation(predicate, w);
00031 }
00032
00041 int variableDenotation(std::string_view variable, const Possibility& p)
00042 {
00043     return p.getAssignment(p.referentSystem->value(variable));
00044 }
00045
00046 }
```

Index