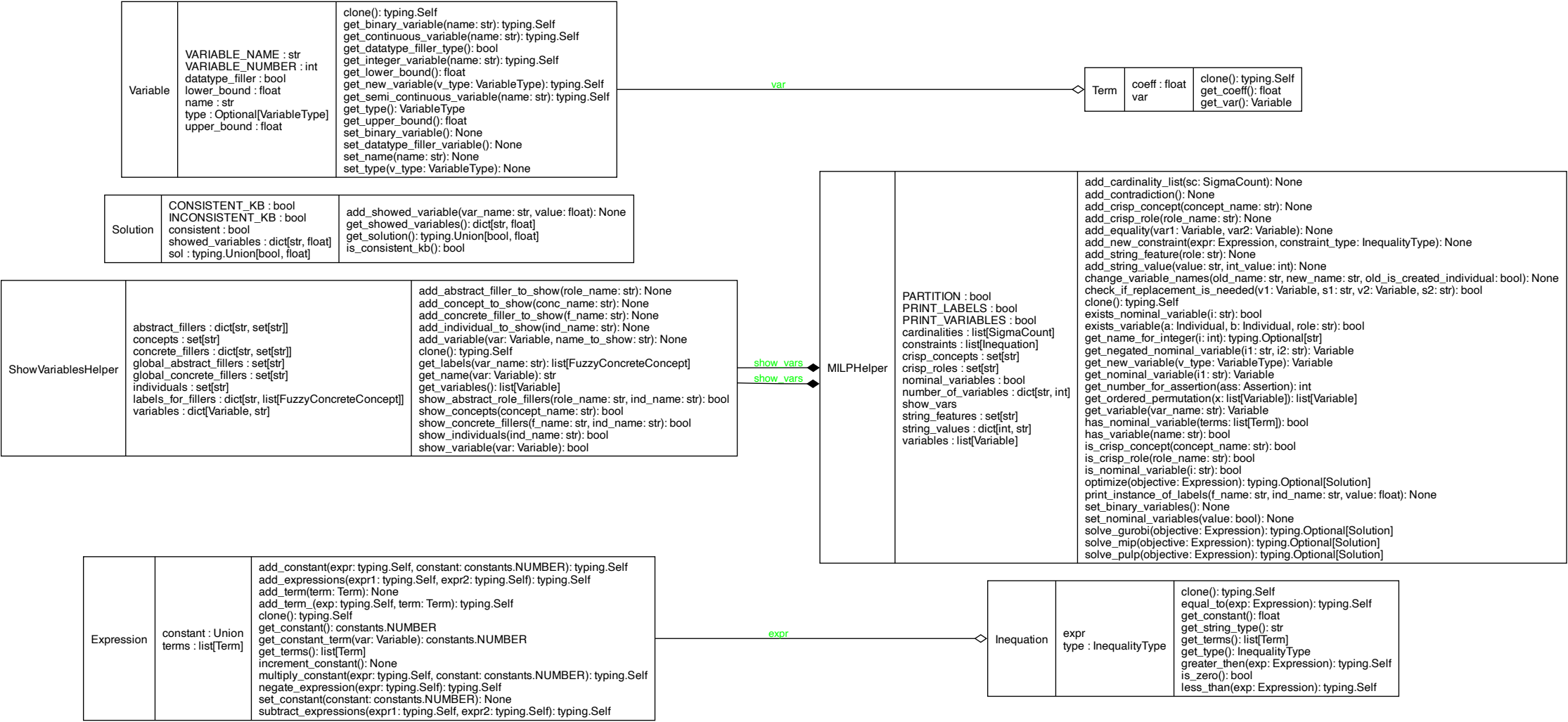
Variable
VARIABLE_NAME : str
VARIABLE_NUMBER : int
datatype_filler : bool
lower_bound : float
name : str
type : Optional[VariableType]
upper_bound : float
clone(): typing.Self
get_binary_variable(name: str): typing.Self
get_continuous_variable(name: str): typing.Self
get_datatype_filler_type(): bool
get_integer_variable(name: str): typing.Self
get_lower_bound(): float
get_new_variable(v_type: VariableType): typing.Self
get_semi_continuous_variable(name: str): typing.Self
get_type(): VariableType
get_upper_bound(): float
set_binary_variable(): None
set_datatype_filler_variable(): None
set_name(name: str): None
set_type(v_type: VariableType): None
var
Term
coeff : float
var
clone(): typing.Self
get_coeff(): float
get_var(): Variable
Solution
CONSISTENT_KB : bool
INCONSISTENT_KB : bool
consistent : bool
showed_variables : dict[str, float]
sol : typing.Union[bool, float]
add_showed_variable(var_name: str, value: float): None
get_showed_variables(): dict[str, float]
get_solution(): typing.Union[bool, float]
is_consistent_kb(): bool
ShowVariablesHelper
abstract_fillers : dict[str, set[str]]
concepts : set[str]
concrete_fillers : dict[str, set[str]]
global_abstract_fillers : set[str]
global_concrete_fillers : set[str]
individuals : set[str]
labels_for_fillers : dict[str, list[FuzzyConcreteConcept]]
variables : dict[Variable, str]
add_abstract_filler_to_show(role_name: str): None
add_concept_to_show(conc_name: str): None
add_concrete_filler_to_show(f_name: str): None
add_individual_to_show(ind_name: str): None
add_variable(var: Variable, name_to_show: str): None
clone(): typing.Self
get_labels(var_name: str): list[FuzzyConcreteConcept]
get_name(var: Variable): str
get_variables(): list[Variable]
show_abstract_role_fillers(role_name: str, ind_name: str): bool
show_concepts(concept_name: str): bool
show_concrete_fillers(f_name: str, ind_name: str): bool
show_individuals(ind_name: str): bool
show_variable(var: Variable): bool
show_vars
show_vars
MILPHelper
PARTITION : bool
PRINT_LABELS : bool
PRINT_VARIABLES : bool
cardinalities : list[SigmaCount]
constraints : list[Inequation]
crisp_concepts : set[str]
crisp_roles : set[str]
nominal_variables : bool
number_of_variables : dict[str, int]
show_vars
string_features : set[str]
string_values : dict[int, str]
variables : list[Variable]
add_cardinality_list(sc: SigmaCount): None
add_contradiction(): None
add_crisp_concept(concept_name: str): None
add_crisp_role(role_name: str): None
add_equality(var1: Variable, var2: Variable): None
add_new_constraint(expr: Expression, constraint_type: InequalityType): None
add_string_feature(role: str): None
add_string_value(value: str, int_value: int): None
change_variable_names(old_name: str, new_name: str, old_is_created_individual: bool): None
check_if_replacement_is_needed(v1: Variable, s1: str, v2: Variable, s2: str): bool
clone(): typing.Self
exists_nominal_variable(i: str): bool
exists_variable(a: Individual, b: Individual, role: str): bool
get_name_for_integer(i: int): typing.Optional[str]
get_negated_nominal_variable(i1: str, i2: str): Variable
get_new_variable(v_type: VariableType): Variable
get_nominal_variable(i1: str): Variable
get_number_for_assertion(ass: Assertion): int
get_ordered_permutation(x: list[Variable]): list[Variable]
get_variable(var_name: str): Variable
has_nominal_variable(terms: list[Term]): bool
has_variable(name: str): bool
is_crisp_concept(concept_name: str): bool
is_crisp_role(role_name: str): bool
is_nominal_variable(i: str): bool
optimize(objective: Expression): typing.Optional[Solution]
print_instance_of_labels(f_name: str, ind_name: str, value: float): None
set_binary_variables(): None
set_nominal_variables(value: bool): None
solve_gurobi(objective: Expression): typing.Optional[Solution]
solve_mip(objective: Expression): typing.Optional[Solution]
solve_pulp(objective: Expression): typing.Optional[Solution]
Expression
constant : Union
terms : list[Term]
add_constant(expr: typing.Self, constant: constants.NUMBER): typing.Self
add_expressions(expr1: typing.Self, expr2: typing.Self): typing.Self
add_term(term: Term): None
add_term_(exp: typing.Self, term: Term): typing.Self
clone(): typing.Self
get_constant(): constants.NUMBER
get_constant_term(var: Variable): constants.NUMBER
get_terms(): list[Term]
increment_constant(): None
multiply_constant(expr: typing.Self, constant: constants.NUMBER): typing.Self
negate_expression(expr: typing.Self): typing.Self
set_constant(constant: constants.NUMBER): None
subtract_expressions(expr1: typing.Self, expr2: typing.Self): typing.Self
expr
Inequation
expr
type : InequalityType
clone(): typing.Self
equal_to(exp: Expression): typing.Self
get_constant(): float
get_string_type(): str
get_terms(): list[Term]
get_type(): InequalityType
greater_then(exp: Expression): typing.Self
is_zero(): bool
less_than(exp: Expression): typing.Self