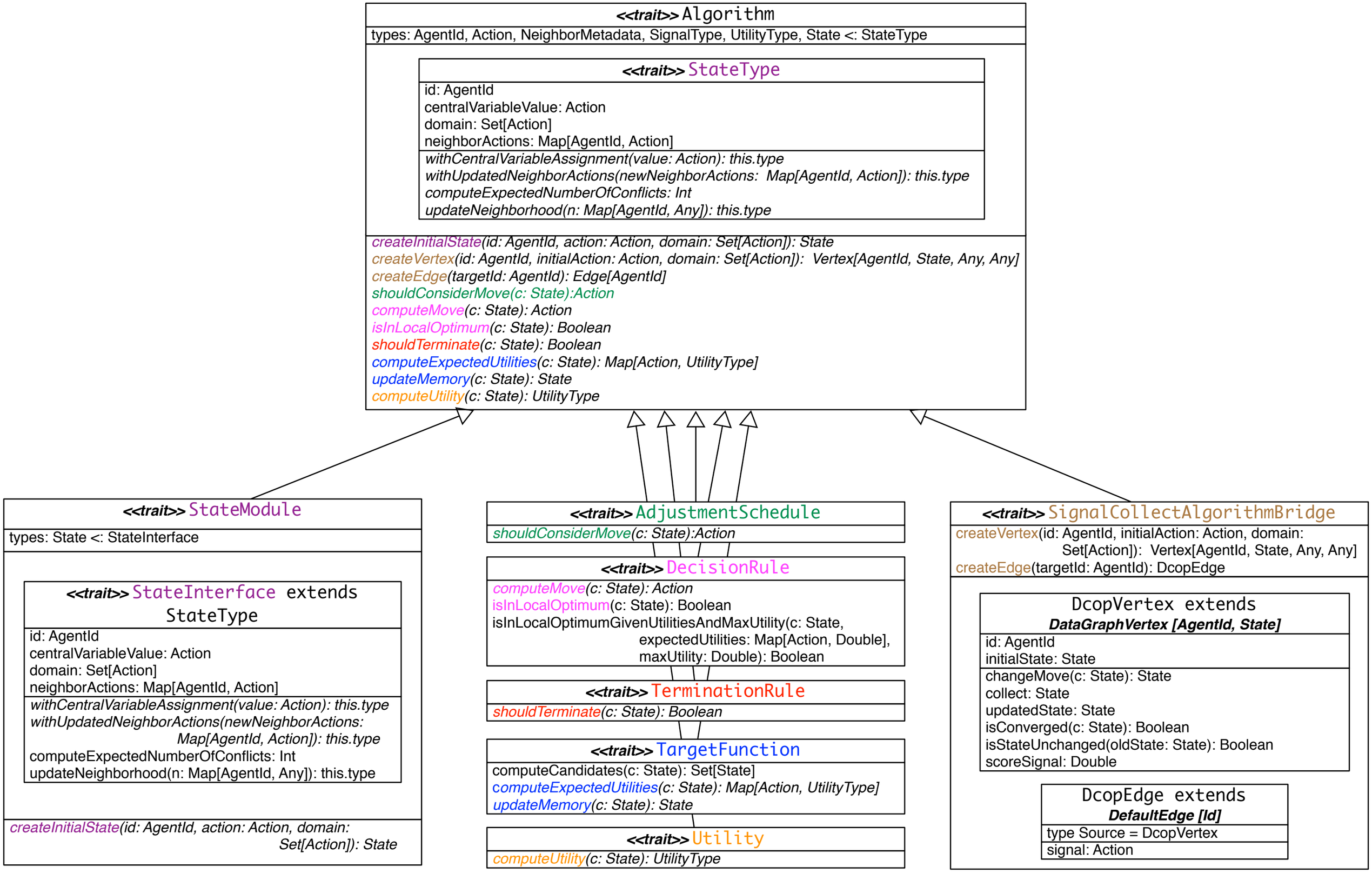
«trait» Algorithm
types: AgentId, Action, NeighborMetadata, SignalType, UtilityType, State <: StateType
«trait» StateType
id: AgentId
centralVariableValue: Action
domain: Set[Action]
neighborActions: Map[AgentId, Action]
withCentralVariableAssignment(value: Action): this.type
withUpdatedNeighborActions(newNeighborActions: Map[AgentId, Action]): this.type
computeExpectedNumberOfConflicts: Int
updateNeighborhood(n: Map[AgentId, Any]): this.type
createInitialState(id: AgentId, action: Action, domain: Set[Action]): State
createVertex(id: AgentId, initialAction: Action, domain: Set[Action]): Vertex[AgentId, State, Any, Any]
createEdge(targetId: AgentId): Edge[AgentId]
shouldConsiderMove(c: State):Action
computeMove(c: State): Action
isInLocalOptimum(c: State): Boolean
shouldTerminate(c: State): Boolean
computeExpectedUtilities(c: State): Map[Action, UtilityType]
updateMemory(c: State): State
computeUtility(c: State): UtilityType
«trait» StateModule
types: State <: StateInterface
«trait» StateInterface extends StateType
id: AgentId
centralVariableValue: Action
domain: Set[Action]
neighborActions: Map[AgentId, Action]
withCentralVariableAssignment(value: Action): this.type
withUpdatedNeighborActions(newNeighborActions: Map[AgentId, Action]): this.type
computeExpectedNumberOfConflicts: Int
updateNeighborhood(n: Map[AgentId, Any]): this.type
createInitialState(id: AgentId, action: Action, domain: Set[Action]): State
«trait» AdjustmentSchedule
shouldConsiderMove(c: State):Action
«trait» DecisionRule
computeMove(c: State): Action
isInLocalOptimum(c: State): Boolean
isInLocalOptimumGivenUtilitiesAndMaxUtility(c: State, expectedUtilities: Map[Action, Double], maxUtility: Double): Boolean
«trait» TerminationRule
shouldTerminate(c: State): Boolean
«trait» TargetFunction
computeCandidates(c: State): Set[State]
computeExpectedUtilities(c: State): Map[Action, UtilityType]
updateMemory(c: State): State
«trait» Utility
computeUtility(c: State): UtilityType
«trait» SignalCollectAlgorithmBridge
createVertex(id: AgentId, initialAction: Action, domain: Set[Action]): Vertex[AgentId, State, Any, Any]
createEdge(targetId: AgentId): DcopEdge
DcopVertex extends DataGraphVertex [AgentId, State]
id: AgentId
initialState: State
changeMove(c: State): State
collect: State
updatedState: State
isConverged(c: State): Boolean
isStateUnchanged(oldState: State): Boolean
scoreSignal: Double
DcopEdge extends DefaultEdge [Id]
type Source = DcopVertex
signal: Action