Appendix 2: Classification Problem Solving

Enrico Motta[1], Wenjin Lu[1]

Abstract

In this appendix we show the complete specification of the method ontology and the problem solving methods which are part of the library of classification components.

Identifier:	**D1_Appx2**
Class:	**Deliverable**
Version:	**1.0**
Date:	**7 September, 2000**
Status:	**Draft**
Distribution:	**Project Members**
Responsible partner:	**The Open University**

```
;;;THE-VIRTUAL-SOLUTION-SPACE
(def-function the-virtual-solution-space (?init-space ?refs)
                                          -> ?solution-space
  "The space generated by refinement application from an initial solution
   space"
  :constraint (and (every ?refs refiner)
                   (solution-space ?init-space))
  :def (= ?solution-space
          (setofall ?sol2 (or (member ?sol2 ?init-space)
                              (exists ?sol3
                                      (and (member ?sol3
                                                   (the-virtual-solution-space
                                                    ?init-space ?refs))
                                           (generalised-refinement-of ?sol2
                                                                      ?sol3
                                                                      ?refs)))))))

;;; CLASS CANDIDATE-EXCLUSION-CRITERION
(def-class candidate-exclusion-criterion ()
  "this provides us with a way to specify when a solution is not
   admissible.  we also use this in conjunction with a solution refinement
   hierarchy, to prune all refinements of an unadmissible solution"
  ((applies-to-match-score-type :type match-score-type)
   (has-candidate-exclusion-relation :type unary-relation)
   ))

;;;INSTANCE DEFAULT-CANDIDATE-EXCLUSION-CRITERION
(def-instance default-candidate-exclusion-criterion candidate-exclusion-criterion
  ((applies-to-match-score-type default-match-score)
   (has-candidate-exclusion-relation default-candidate-exclusion-relation)))

;;;RELATION DEFAULT-CANDIDATE-EXCLUSION-RELATION
(def-relation default-candidate-exclusion-relation (?score)
  "a solution is excluded if it has one or more inconsistent features"
  :constraint (default-match-score ?score)
  :iff-def (> (length (first ?score)) 0))

;;;RELATION RULED-OUT-SOLUTION
(def-relation ruled-out-solution (?sol ?score ?criterion)
  :iff-def (holds (the ?rel (has-candidate-exclusion-relation  ?criterion ?rel))
                  ?score))

;;;AXIOM EXCLUSION-IS-MONOTONIC
(def-axiom exclusion-is-monotonic
  "This axiom states that the exclusion  criterion is monotonic. That is,
   if a solution, ?sol, is ruled out, then any solution which has a worse score
than
   ?sol will also be ruled out"
  (forall (?sol1 ?sol2 ?obs ?criterion)
          (=> (and (ruled-out-solution
                    ?sol1 (apply-match-criterion ?criterion ?obs ?sol1) ?criterion)
                   (not (better-match-than ?sol2 ?sol1 ?obs ?criterion)))
              (ruled-out-solution
               ?sol2 (apply-match-criterion ?criterion ?obs ?sol2) ?criterion))))
```

```
(def-axiom CONGRUENT-ADMISSIBILITY-AND-EXCLUSION-CRITERIA
  "This axiom states that the admissibility and exclusion criteria
   have to be congruent: no solution should ever satisfy both"
  (forall (?sol ?task)
          (=> (member ?sol
                      (the-virtual-solution-space
                       (role-value ?task has-candidate-solutions)
                       (role-value ?task has-refiners)))
              (not (and (admissible-solution
                         ?sol
                         (apply-match-criterion
                          (role-value ?task 'has-match-criterion)
                          (role-value ?task 'has-observables)
                          ?sol)
                         (role-value
                          ?task
                          'has-solution-admissibility-criterion))
                        (ruled-out-solution ?sol
                                            (apply-match-criterion
                                             (role-value ?task 'has-match-
criterion)
                                             (role-value ?task 'has-observables)
                                             ?sol)
                                            (role-value
                                             ?psm 'has-solution-exclusion-
criterion)))))))
```

```
;;;HEURISTIC-ADMISSIBLE-SOL-CLASSIFIER
(def-class heuristic-admissible-sol-classifier (problem-solving-method) ?psm
  "This is a basic method for finding an admissible solution to a
   classification problem.  It uses a backtracking hill-climbing solution to
   home in on the most promising solutions.  It is a heuristic, hierarchical
   classifier, as it provides for both abstraction and refinement mechanisms.
   If no refinement mechanisms are provided, then the psm just reduces to a
   'flat' classifier."

  ((has-input-role :value has-abstractors
                   :value has-refiners
                   :value has-observables
                   :value has-candidate-solutions
                   :value has-solution-exclusion-criterion
                   )

   (has-output-role :value has-solution)
   (has-abstractors :type abstractors)
   (has-refiners :type refiners)
   (has-observables :type observables)
   (has-solution-exclusion-criterion :type candidate-exclusion-criterion
                                     :default-value default-candidate-exclusion-
criterion)
   (has-candidate-solutions :type solution-space)
   (has-solution :type solution)
   (has-assumption
    :value (kappa (?psm)
                  (not (exists (?ob1 ?ob2)
                               (and (or (member ?ob1
                                                (role-value ?psm has-observables))
                                        (exists ?ob
                                                (and (member
                                                      ?ob
                                                      (role-value ?psm has-
observables))
                                                     (generalised-abstract-link
                                                      ?ob1 ?ob
                                                      (role-value ?psm 'has-
abstractors)))))
                                    (generalised-abstract-link
                                     ?ob2 ?ob1
                                     (role-value ?psm 'has-abstractors))
                                    (generalised-abstract-link
                                     ?ob1 ?ob2
                                     (role-value ?psm 'has-abstractors))))))
    :value (kappa (?psm)
                  (not (exists (?sol1 ?sol2)
                               (and (or (member ?sol1
                                                (role-value ?psm 'has-candidate-
solutions))
                                        (exists ?sol
                                                (and (member ?sol
                                                             (role-value
                                                              ?psm
                                                              'has-candidate-
solutions))
                                                     (generalised-refinement-of
                                                      ?sol1 ?sol
                                                      (role-value ?psm has-
refiners)))))
                                    (generalised-refinement-of
                                     ?sol2 ?sol1
                                     (role-value ?psm has-refiners))
                                    (generalised-refinement-of
                                     ?sol1 ?sol2
                                     (role-value ?psm has-refiners))))))




    :value (kappa (?psm)
                  (exists ?sol
                          (and (admissible-solution
                                ?sol
                                (apply-match-criterion
                                 (role-value ?psm 'has-match-criterion)
                                 (role-value ?psm 'has-observables)
                                 ?sol)
                                (role-value
                                 ?psm
                                 'has-solution-admissibility-criterion))
                               (member ?sol
                                       (the-virtual-solution-space
                                        (role-value ?psm has-candidate-solutions)
                                        (role-value ?psm has-refiners))))))
    :value (kappa (?psm)
                  (forall (?sol ?score)
                          (=>
                           (and
                            (sol-has-match-score ?sol
                                                 (role-value ?psm 'has-observables)
                                                 ?score
                                                 (role-value
                                                  ?psm 'has-match-criterion))
                            (ruled-out-solution ?sol ?score
                                                (role-value
                                                 ?psm 'has-solution-exclusion-
criterion)))
                           (not (exists
                                 ?sol2
                                 (and (generalised-refinement-of
                                       ?sol2 ?sol (role-value
                                                   ?psm has-refiners))
                                      (admissible-solution
                                       ?sol2
                                       (apply-match-criterion
                                        (role-value ?psm 'has-match-criterion)
                                        (role-value ?psm 'has-observables)
                                        ?sol2)
                                       (role-value
                                        ?psm
                                        'has-solution-admissibility-
criterion))))))))

    :documentation
```

```
          "There are 4 assumptions associated with this method.  The first
           two state that the abstraction and refinement hierarchies have to
           be free of cycles.  The third states that the psm assumes the existence
           of a solution in the virtual solution space - i.e., if such a solution
           exists, the psm will find it.  The final assumption states that the
           method assumes that the exclusion criterion is correct.  That is, if a
           solution is excluded, all its refinements can be excluded too.")
   (has-postcondition
    :value (kappa (?psm ?sol)
                  (admissible-solution
                   ?sol
                   (apply-match-criterion
                    (role-value ?psm 'has-match-criterion)
                    (role-value ?psm 'has-observables)
                    ?sol)
                   (role-value
                    ?psm
                    'has-solution-admissibility-criterion)))
    :documentation "The output solution satisfies the admissibility criterion")

   (has-body
    :value
    '(lambda (?psm)
       (in-environment
        ((?obs . (achieve-generic-subtask
                  ?psm abstraction
                  'has-observables (role-value ?psm has-observables)
                  'has-abstractors (role-value ?psm has-abstractors))))

        (admissible-solution-search ?psm
                                    ?obs
                                    (role-value ?psm has-refiners)
                                    (role-value ?psm has-candidate-solutions)
                                    (role-value ?psm
                                                has-match-criterion)
                                    (role-value ?psm
                                                has-solution-admissibility-
criterion)
                                    (role-value
                                     ?psm
                                     has-solution-exclusion-criterion)))))

   (has-output-mapping
    :value '(lambda (?psm ?result)
              (list-of ?result))))

  :own-slots ((tackles-task-type single-solution-classification-task)
              (has-generic-subtasks '(abstraction
                                      refinement rank-solutions))))
```

```
(def-procedure admissible-solution-search (?psm ?obs ?refs
                                                ?candidates
                                                ?match-criterion
                                                ?admissibility-criterion
                                                ?exclusion-criterion)

  :constraint (and (solution-space ?candidates)
                   (list ?refs)
                   (observables ?obs)
                   (problem-solving-method ?psm)
                   (match-criterion ?match-criterion)
                   (admissibility-criterion ?admissibility-criterion)
                   (solution-admissibility-criterion ?admissibility-criterion)
                   (solution-exclusion-criterion ?exclusion-criterion)
                   )
  :body (do
          (if (exists ?sol
                      (and (member ?sol ?candidates)
                           (admissible-solution
                            ?sol
                            (apply-match-criterion
                             ?match-criterion
                             ?obs
                             ?sol)
                            ?admissibility-criterion)))
            ?sol
            (in-environment
             ((?ranked-candidates . (achieve-generic-subtask
                                     ?psm rank-solutions
                                     'has-observables ?obs
                                     'has-candidate-solutions ?candidates

                                     'has-match-criterion ?match-criterion)))
             (loop for ?candidate in ?ranked-candidates
                   do
                   (if (not (ruled-out-solution ?candidate
                                                (apply-match-criterion ?match-
criterion
                                                                       ?obs
                                                                       ?candidate)
                                                ?exclusion-criterion))
                     (in-environment
                      ((?refined-sols . (achieve-generic-subtask
                                         ?psm refinement
                                         'has-candidate-solution ?candidate
                                         'has-observables ?obs
                                         'has-refiners ?refs)))
                      (if (not (null ?refined-sols))
                        (in-environment
                         ((?result . (admissible-solution-search
                                      ?psm ?obs ?refs
                                      ?refined-sols
                                      ?match-criterion
                                      ?admissibility-criterion
                                      ?exclusion-criterion)))
                         (if (not (= ?result :nothing))
                           (return ?result))))))))))
```

```
;;;HEURISTIC-OPTIMAL-SOL-CLASSIFIER
(def-class heuristic-optimal-sol-classifier (problem-solving-method) ?psm
  "This method exhaustively searches a solution space looking for an optimal
    solution.  It uses the exclusion criterion to prune the search space.  It
   returns one or more optimal solutions"

  ((has-input-role :value has-abstractors
                   :value has-refiners
                   :value has-observables
                   :value has-candidate-solutions
                   :value has-solution-exclusion-criterion
                   )
   (has-control-role :value 'has-current-solutions
                     :value 'has-ranked-candidates)
   (has-output-role :value has-solutions)
   (has-current-solutions :type solution-space)
   (has-ranked-candidates :type solution-space)
   (has-abstractors :type abstractors)
   (has-refiners :type refiners)
   (has-observables :type observables)
   (has-solution-exclusion-criterion :type candidate-exclusion-criterion
                                     :default-value default-candidate-exclusion-
criterion)
   (has-candidate-solutions :type solution-space)
   (has-solutions :type solution-space)
   (has-assumption
    :value (kappa (?psm)
                  (not (exists (?ob1 ?ob2)
                               (and (or (member ?ob1
                                                (role-value ?psm has-observables))
                                        (exists ?ob
                                                (and (member
                                                      ?ob
                                                      (role-value ?psm has-
observables))
                                                     (generalised-abstract-link
                                                      ?ob1 ?ob
                                                      (role-value ?psm 'has-
abstractors)))))
                                    (generalised-abstract-link
                                     ?ob2 ?ob1
                                     (role-value ?psm 'has-abstractors))
                                    (generalised-abstract-link
                                     ?ob1 ?ob2
                                     (role-value ?psm 'has-abstractors))))))
    :value (kappa (?psm)
                  (not (exists (?sol1 ?sol2)
                               (and (or (member ?sol1
                                                (role-value ?psm 'has-candidate-
solutions))
                                        (exists ?sol
                                                (and (member ?sol
                                                             (role-value
                                                              ?psm
                                                              'has-candidate-
solutions))
                                                     (generalised-refinement-of
                                                      ?sol1 ?sol
                                                (role-value ?psm has-
refiners)))))
                                    (generalised-refinement-of
                                     ?sol2 ?sol1
                                     (role-value ?psm has-refiners))
                                    (generalised-refinement-of
                                     ?sol1 ?sol2
                                     (role-value ?psm has-refiners))))))

    :value (kappa (?psm)
                  (exists ?sol
                          (and (admissible-solution
                                ?sol
                                (apply-match-criterion
                                 (role-value ?psm 'has-match-criterion)
                                 (role-value ?psm 'has-observables)
                                 ?sol)
                                (role-value
                                 ?psm
                                 'has-solution-admissibility-criterion))
                               (member ?sol
                                       (the-virtual-solution-space
                                        (role-value ?psm has-candidate-solutions)
                                        (role-value ?psm has-refiners))))))
    :value (kappa (?psm)
                  (forall (?sol ?score)
                          (=>
                           (and
                            (sol-has-match-score ?sol
                                                 (role-value ?psm 'has-observables)
                                                 ?score
                                                 (role-value
                                                  ?psm 'has-match-criterion))
                            (ruled-out-solution ?sol ?score
                                                (role-value
                                                 ?psm 'has-solution-exclusion-
criterion)))
                           (not (exists
                                 ?sol2
                                 (and (generalised-refinement-of
                                       ?sol2 ?sol (role-value
                                                   ?psm has-refiners))
                                      (admissible-solution
                                       ?sol2
                                       (apply-match-criterion
                                        (role-value ?psm 'has-match-criterion)
                                        (role-value ?psm 'has-observables)
                                        ?sol2)
                                       (role-value
                                        ?psm
                                        'has-solution-admissibility-
criterion))))))))
    :documentation
       "There are 4 assumptions associated with this method.  The first
        two state that the abstraction and refinement hierarchies have to be free
        of cycles.  The third states that the psm assumes the
        existence of a solution in the virtual solution space - i.e., if such
        a solution exists, the psm will find it.  The final assumption states
```

```
         that the method assumes that the exclusion criterion is correct.
         That is, if a solution is excluded, all its refinements can be
         excluded too.")
     (has-postcondition
      :value (kappa (?psm ?sols)
                    (forall ?sol
                            (=> (member ?sol ?sols)
                                (and

                                 (admissible-solution
                                  ?sol
                                  (apply-match-criterion
                                   (role-value ?psm 'has-match-criterion)
                                   (role-value ?psm 'has-observables)
                                   ?sol)
                                  (role-value
                                   ?psm
                                   'has-solution-admissibility-criterion))

                                 (best-match
                                  (role-value ?task 'has-observables)
                                  ?sol
                                  (the-virtual-solution-space
                                   (role-value ?psm has-observables)
                                   (role-value ?psm 'has-candidate-solutions))
                                  (role-value ?task 'has-match-criterion))))))
      :documentation "The output solutions are both admissible and optimal
                      with respect to the solution space generated through
                      refinement application")

     (has-body
      :value
      '(lambda (?psm)
         (in-environment
          ((?obs . (achieve-generic-subtask
                    ?psm abstraction
                    'has-observables (role-value ?psm has-observables)
                    'has-abstractors (role-value ?psm has-abstractors))))
          (do
            (set-role-value ?psm 'has-ranked-candidates
                            (achieve-generic-subtask
                             ?psm rank-solutions
                             'has-observables ?obs
                             'has-candidate-solutions
                             (role-value ?psm 'has-candidate-solutions)
                             'has-match-criterion  (role-value
                                                    ?psm has-match-criterion)))

            ;;Initialize current solutions
            (set-role-value ?psm 'has-current-solutions nil)
            (if (admissible-solution
                 (first (role-value ?psm 'has-ranked-candidates))
                 (apply-match-criterion  (role-value ?psm has-match-criterion)
                                         ?obs
                                         (first
                                          (role-value ?psm 'has-ranked-candidates)))
                 (role-value ?psm
                             'has-solution-admissibility-criterion))
              (do
                (set-role-value ?psm 'has-current-solutions
                                (List-of (first
                                          (role-value ?psm 'has-ranked-candidates))))
                (loop for ?candidate in (rest (role-value ?psm 'has-ranked-
  candidates))
                      do
                      (if (better-match-than
                           (first (role-value ?psm 'has-ranked-candidates))
                           ?candidate ?obs
                           (role-value ?psm has-match-criterion))
                        (return :nothing)
                        (set-role-value ?psm 'has-current-solutions
                                        (cons ?candidate
                                              (role-value ?psm
                                                          'has-current-
  solutions)))))))
            ;;remove ruled out candidates
            (loop for ?candidate in (reverse (role-value ?psm 'has-ranked-
  candidates))
                  do
                  (if (ruled-out-solution
                       ?candidate (apply-match-criterion
                                   (role-value ?psm has-match-criterion)
                                   ?obs
                                   ?candidate)
                       (role-value ?psm
                                   'has-solution-exclusion-criterion))
                    (set-role-value ?psm 'has-ranked-candidates
                                    (remove ?candidate
                                            (role-value ?psm 'has-ranked-
  candidates)))
                    (return :nothing)))

            (if (null (role-value ?psm 'has-ranked-candidates))
              (role-value ?psm
                          'has-current-solutions)
              (complete-optimal-search ?psm
                                       ?obs
                                       (role-value ?psm has-refiners)
                                       (first (role-value ?psm 'has-ranked-
  candidates))
                                       (rest (role-value ?psm 'has-ranked-
  candidates))
                                       (role-value ?psm
                                                   'has-current-solutions)
                                       (role-value ?psm has-match-criterion)
                                       (role-value  ?psm
                                                    'has-solution-admissibility-
  criterion)
                                       (role-value  ?psm
                                                    'has-solution-exclusion-
  criterion)))))))

    :own-slots ((tackles-task-type optimal-classification-task)
                (has-generic-subtasks '(abstraction
                                        refinement rank-solutions))))
```

```
(def-procedure complete-optimal-search (?psm ?obs ?refs
                                          ?candidate
                                          ?other-candidates
                                          ?current-solutions
                                          ?match-criterion
                                          ?admissibility-criterion
                                          ?exclusion-criterion)
  "Neither ?candidate no ?other-candidates have been ruled out
   by the exclusion criterion"
  :constraint (and (solution ?candidate)
                   (list ?other-candidates)
                   (list ?refs)
                   (list ?current-solutions)
                   (observables ?obs)
                   (problem-solving-method ?psm)
                   (match-criterion ?match-criterion)
                   (admissibility-criterion ?admissibility-criterion)
                   (solution-admissibility-criterion ?admissibility-criterion)
                   (solution-exclusion-criterion ?exclusion-criterion)
                   (not (ruled-out-solution
                         ?candidate (apply-match-criterion
                                     ?match-criterion
                                     ?obs
                                     ?candidate)
                         ?exclusion-criterion))
                   (forall ?c
                           (=> (member ?c ?other-candidates)
                               (not (ruled-out-solution
                                     ?c (apply-match-criterion
                                         ?match-criterion
                                         ?obs
                                         ?c)
                                     ?exclusion-criterion)))))
  :body (in-environment
         ((?good-refined-sols . (filter (achieve-generic-subtask
                                         ?psm refinement
                                         'has-candidate-solution ?candidate
                                         'has-observables ?obs
                                         'has-refiners ?refs)
                                        '(kappa (?sol)
                                          (not (ruled-out-solution
                                                ?sol (apply-match-criterion
                                                      ?match-criterion
                                                      ?obs
                                                      ?sol)
                                                ?exclusion-criterion))))))
         (if (null ?good-refined-sols)
           (if (null ?other-candidates)
             ?current-solutions
             (complete-optimal-search ?psm ?obs ?refs
                                      (first ?other-candidates)
                                      (rest ?other-candidates)
                                      ?current-solutions
                                      ?match-criterion
                                      ?admissibility-criterion
                                      ?exclusion-criterion))

           ;;there are some new useful refinements
           (in-environment
            ((?ranked-candidates . (achieve-generic-subtask
                                    ?psm rank-solutions
                                    'has-observables ?obs
                                    'has-candidate-solutions (append
                                                              ?other-candidates
                                                              ?good-refined-sols)
                                    'has-match-criterion  ?match-criterion))
             (?new-admissible-solutions . (filter ?good-refined-sols
                                                  '(kappa (?sol)
                                                    (admissible-solution
                                                     ?sol
                                                     (apply-match-criterion
                                                      ?match-criterion ?obs ?sol)
                                                     ?admissibility-criterion))))
             (?new-current-solutions . (if (null ?new-admissible-solutions)
                                         ?current-solutions
                                         (in-environment
                                          ((?ranked-sols . (achieve-generic-subtask
                                                            ?psm rank-solutions
                                                            'has-observables ?obs
                                                            'has-candidate-
solutions
                                                            (append
                                                             ?current-solutions
                                                             ?new-admissible-
solutions)
                                                            'has-match-criterion
                                                            ?match-criterion)))
                                          (cons (first ?ranked-sols)
                                                (filter (rest ?ranked-sols)
                                                        '(kappa (?sol)
                                                          (not (better-match-than
                                                                (first ?ranked-
sols)
                                                                ?sol
                                                                ?obs
                                                                ?match-
criterion))))))))
            (complete-optimal-search ?psm ?obs ?refs
                                     (first ?ranked-candidates)
                                     (rest ?ranked-candidates)
                                     ?new-current-solutions
                                     ?match-criterion
                                     ?admissibility-criterion
                                     ?exclusion-criterion)))))


;;;;;;;;;;;;;;;;;;;; Beginning of abstraction ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(def-class abstractor (function)
  ((domain :value observables)
   (range :value observable)
   (applicability-condition :type abstractor-applicability-condition-class)))
```

```
(def-class abstractor-applicability-condition-class (unary-relation) ?x
  "The applicability condition for an abstractor must be a relation whose domain is
    a subset of observables"
  :iff-def (subset (the ?d (domain ?x ?d))
                   observables))

(def-function apply-abstract-operator (?ab ?observables-in) -> ?ob
  :constraint (and (abstractor ?ab)
                   (observable ?ob)
                   (observables ?observables-in)
                   )
  :body (if (abstractor-is-applicable? ?ab ?observables-in)
          (call (the ?body (has-body ?ab ?body))
                ?observables-in)))

(def-relation abstractor-is-applicable? (?ab ?observables)
  :constraint (and (abstractor ?ab)
                   (observables ?observables))
  :iff-def (holds (the ?appl (applicability-condition ?ab ?appl)) ?observables))

(def-class abstraction (goal-specification-task) ?task
  "The goal of this task is to abstract from the given observables, using the given
   abstractors.  Note that the output also includes the original (non-abstracted)
   observables"
  ((has-input-role :value has-observables
                   :value has-abstractors)
   (has-output-role :value has-abstract-observables)
   (has-observables :type observables)
   (has-abstract-observables :type observables)
   (has-abstractors :type list)
   (has-goal-expression
    :value (kappa (?task ?observables)
                  (and (forall ?ob
                               (=> (member ?ob ?observables)
                                   (or (member ?ob (role-value ?task has-
observables))
                                       (generalized-abstract-from
                                        ?observables
                                        (role-value ?task has-observables)
                                        (role-value ?task has-abstractors)))))
                       (not (exists (?ab ?new-ob)
                                    (and (member ?ab (role-value
                                                      ?task has-abstractors))
                                         (abstractor-is-applicable?
                                          ?ab ?observables)
                                         (= ?new-ob (apply-abstract-operator
                                                     ?ab ?observables))
                                         (not (member ?new-ob
?observables)))))))))
```

```
(def-relation generalized-abstract-from (?observables-out ?observables-in ?abs)
  :constraint (and (observables ?observables-out)
                   (observables ?observables-in)
                   (abstractors ?abs))

  :iff-def (forall (?ob)
                   (=> (member ?ob ?observables-out)
                       (or (directly-abstracted-from ?ob ?observables-in ?abs)
                           (exists (?observables-temp)
                                   (and (directly-abstracted-from
                                         ?ob ?observables-temp ?abs)
                                        (forall (?ob-temp)
                                                (=> (member ?ob-temp ?observables-
temp)
                                                    (or (member ?ob-temp
?observables-in)
                                                        (observable-abstracted-from
                                                         ?ob-temp ?observables-in
                                                         ?abs))))))))))

(def-relation generalised-abstract-link (?ob1 ?ob2 ?abs)
  "?ob1 is in a chain of abstraction which stems from ?ob2"
  :iff-def (and (member ?ob1 ?observables1)
                (member ?ob2 ?observables2)
                (generalized-abstract-from ?observables1 ?observables2 ?abs)))



(def-relation observable-abstracted-from (?ob ?observables ?abs)
  :iff-def (or (directly-abstracted-from ?ob ?observables ?abs)
               (exists (?observables2)
                       (and (generalized-abstract-from ?observables2 ?observables
?abs)
                            (member ?ob ?observables2)))))

(def-relation directly-abstracted-from (?ob ?observables ?abs)
  :constraint (and (observables ?observables)
                   (observable ?ob)
                   (abstractors ?abs))
  :iff-def (exists (?ab)
                   (and (member ?ab ?abs)
                        (abstractor-is-applicable? ?ab ?observables)
                        (member ?ob (apply-abstract-operator ?ab ?observables)))))
```

```
(def-class abstraction-psm (primitive-method) ?psm
  "This method applies abstractors to the given data space (observables) repeatedly
  until no abstractor can be used and returns a data space in which
  all possible abstractions are done"

  ((has-control-role :value has-current-observables
                     :value has-current-abstractors)
   (has-current-observables :type observables)
   (has-current-abstractors :type list)
   (has-body
    :value  '(lambda (?psm)
              (do
                (set-role-value ?psm
                                'has-current-observables
                                (role-value ?psm 'has-observables))
                (set-role-value ?psm 'has-current-abstractors
                                (role-value ?psm 'has-abstractors))
                (repeat
                 (in-environment
                  ((?ab . (achieve-generic-subtask
                           ?psm select-abstractor
                           'has-observables
                           (role-value
                            ?psm 'has-current-observables)
                           'has-abstractors
                           (role-value
                            ?psm 'has-current-abstractors))))
                  (if (abstractor ?ab)
                    (do
                      (set-role-value
                       ?psm 'has-current-abstractors
                       (remove ?ab
                               (role-value
                                ?psm
                                'has-current-abstractors)))
                      (in-environment
                       ((?obs . (the ?obs2 (has-current-observables ?psm ?obs2)))
                        (?ob . (achieve-generic-subtask ?psm one-step-abstraction
                                                        'has-abstractor ?ab
                                                        'has-observables ?obs)))
                       (if (and (observable ?ob)
                                (not (member ?ob ?obs)))
                         (set-role-value ?psm 'has-current-observables
                                         (cons ?ob ?obs)))))
                    (return (role-value ?psm 'has-current-observables))))))))
  :own-slots ((tackles-task-type 'abstraction)))
```

```
(def-class select-abstractor (goal-specification-task) ?task
  ((has-input-role :value has-observables
                   :value has-abstractors)
   (has-output-role :value has-abstractor)
   (has-observables :type observables)
   (has-abstractors  :type list)
   (has-abstractor :type abstractor)
   (has-goal-expression
    :value
    (kappa (?task ?abstractor)
           (and (member ?abstractor (role-value ?task 'has-abstractors))
                (abstractor-is-applicable?
                 ?abstractor (role-value ?task 'has-observables))
                (= ?ob (apply-abstract-operator
                        ?abstractor (role-value ?task 'has-observables)))
                (not (member ?ob (role-value ?task 'has-observables))))))))

(def-class select-abstractor-psm (primitive-method) ?psm
  ((has-body :value '(lambda (?psm)
                       (the ?x
                         (holds (the ?exp (has-goal-expression
                                           (the ?task (tackles-task ?psm ?task))
                                           ?exp))
                                ?psm
                                ?x)))))
  :own-slots ((tackles-task-type 'select-abstractor)))

(def-class one-step-abstraction (goal-specification-task) ?task
  ((has-input-role :value has-observables
                   :value has-abstractor)
   (has-output-role :value has-abstract-observable)
   (has-observables :type observables)
   (has-abstractor  :type abstractor)
   (has-abstract-observable :type observable)
   (has-goal-expression
    :value (kappa (?task ?observable)
                  (directly-abstracted-from ?observable
                                            (role-value ?task 'has-observables)
                                            (list (role-value ?task
                                                              'has-
abstractor)))))))

(def-class one-step-abstraction-psm (primitive-method) ?psm
  ((has-body :value '(lambda (?psm)
                       (apply-abstract-operator
                        (role-value ?psm 'has-abstractor)
                        (role-value ?psm 'has-observables)))))
  :own-slots ((tackles-task-type 'one-step-abstraction)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;; end of abstraction  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;RANK-SOLUTIONS
(def-class rank-solutions (goal-specification-task) ?task
  ((has-input-role :value has-candidate-solutions
                   :value has-observables
                   :value has-match-criterion)
   (has-output-role :value has-solutions)
   (has-candidate-solutions :type solution-space)
   (has-observables :type observables)
   (has-match-criterion :type match-criterion
                        :default-value default-match-criterion)
   (has-solutions :type solution-space)
   (has-goal-expression
    :documentation
    "The goal is to rank the classes according to the match criterion.
     The output should be a list of solutions, in which no solution follows
     one which is worse"
    :value (kappa (?task ?solutions)
                  (forall (?sol1 ?sol2)
                          (=> (and (member ?sol1 ?solutions)
                                   (member ?sol2 ?solutions)
                                   (precedes ?sol1 ?sol2 ?solutions))
                              (not (better-match-score ?sol2 ?sol1))))))))
```

```
(def-class  rank-solutions-psm (primitive-method) ?psm

  ((has-input-role :value has-candidate-solutions
                   :value has-observables
                   :value has-match-criterion)
   (has-output-role :value has-ranked-solutions)
   (has-control-role :value has-sol-score-pairs)
   (has-sol-score-pairs :type list
                        :default-value nil)
   (has-candidate-solutions
    :type solution-space)
   (has-observables :type observables)
   (has-match-criterion :type match-criterion
                        :default-value default-match-criterion)
   (has-ranked-solutions :type list)
   (has-precondition
    :documentation "A list of candidates is required as input"
    :value (kappa (?psm)
                  (list (role-value
                         ?psm 'has-candidate-solutions))))
   (has-postcondition
    :value (kappa (?psm ?solutions)
                  (forall (?sol1 ?sol2)
                          (=> (and (member ?sol1 ?solutions)
                                   (member ?sol2 ?solutions)
                                   (precedes ?sol1 ?sol2 ?solutions))
                              (not (better-match-score ?sol2 ?sol1))))))
   (has-body
    :value '(lambda (?psm)
              (do
                (loop for ?candidate in (role-value
                                         ?psm 'has-candidate-solutions)
                      do
                      (set-role-value
                       ?psm has-sol-score-pairs
                       (cons (list-of ?candidate
                                      (achieve-generic-subtask
                                       ?psm basic-heuristic-match
                                       'has-observables
                                       (role-value
                                        ?psm
                                        has-observables )
                                       'has-candidate-solution ?candidate
                                       'has-match-criterion
                                       (role-value
                                        ?psm
                                        has-match-criterion)))
                             (role-value  ?psm has-sol-score-pairs))))
                (map '(lambda (?pair)
                        (first ?pair))
                     (sort (role-value  ?psm has-sol-score-pairs)
                           '(kappa (?pair1 ?pair2)
                             (better-match-score  (second ?pair1)(second ?pair2)
                                                  (role-value
                                                   ?psm
                                                   'has-match-criterion))))))))
  :own-slots ((has-generic-subtasks '(basic-heuristic-match))
              (tackles-task-type 'rank-solutions)))
```

```
(def-class basic-heuristic-match (goal-specification-task) ?task

  ((has-input-role :value has-candidate-solution
                   :value has-observables
                   :value has-match-criterion)
   (has-output-role :value has-score)
   (has-candidate-solution :type solution)
   (has-observables :type observables)
   (has-match-criterion :type match-criterion
                        :default-value default-match-criterion)
   (has-score :type match-score)
   (has-goal-expression
    ;;the goal is to find the best matching classes
    :value (kappa (?task ?score)
                  (match-score ?score)))))

(def-class  basic-heuristic-match-psm (primitive-method) ?psm

  ((has-precondition
    :documentation "A list of candidates is required as input"
    :value (kappa (?psm)
                  (and (solution (role-value
                                  ?psm 'has-candidate-solution))
                       (exists ?x
                               (member ?X (role-value
                                           ?psm 'has-observables))))))

   (has-postcondition
    :value (kappa (?psm ?score)
                  (match-score ?score)))
   (has-body
    :value '(lambda (?psm)
              (apply-match-criterion
               (role-value
                ?psm 'has-match-criterion)
               (role-value
                ?psm 'has-observables)
               (role-value
                ?psm 'has-candidate-solution)))))
  :own-slots (
              (tackles-task-type 'basic-heuristic-match)))

;;;;;;;;;REFINEMENT TASK;;;;;;;;;;;;;;;;;;;;;;;;

;;;CLASS REFINER
(def-class refiner (function)
  ((domain :value solution)
   (range :value solution-space)
   (applicability-condition :type refiner-applicability-condition-class)))

(def-class refiner-applicability-condition-class (unary-relation) ?x
  "The applicability condition for a refiner must be a relation whose domain is
    a subset of class solution"
  :iff-def (subset ?x solution))
```

```
(def-relation refiner-is-applicable? (?ref ?solution)
  :constraint (and (solution ?solution)
                   (refiner ?ref))
  :iff-def (holds (the ?appl (applicability-condition ?ref ?appl)) ?solution))

(def-relation some-refiner-is-applicable? (?refs ?solution)
  :constraint (and (solution ?solution)
                   (list ?refs))
  :iff-def (exists ?ref (and (member ?ref ?refs)
                             (refiner-is-applicable? ?ref ?solution))))

(def-function apply-refiner-operator (?ref ?solution) -> ?sols
  :constraint (and (solution-space ?sols)
                   (refiner ?ref)
                   (solution ?solution)
                   )
  :body (if (refiner-is-applicable? ?ref ?solution)
          (call (the ?body (has-body ?ref ?body))
                ?solution)))

;;;GENERALISED-REFINEMENT-OF
(def-relation generalised-refinement-of (?sol-out ?sol-in ?refs)
  :iff-def (or (exists ?ref (and (member ?ref ?refs)
                                 (member ?sol-out (apply-refiner-operator
                                                   ?ref ?sol-in))))
               (exists ?sol-temp
                       (and (generalised-refinement-of ?sol-temp ?sol-in ?refs)
                            (generalised-refinement-of ?sol-out ?sol-temp
?refs)))))
```

```
;;;REFINEMENT-THROUGH-SUBCLASS-OF-LINKS
(def-instance refinement-through-subclass-of-links refiner
  "If the solution space is specified by means of classes arranged in a subclass-of
   hierarchy, then this is a good refiner to use"
  ((has-body '(lambda (?sol)
                (setofall ?sub (direct-subclass-of ?sub ?sol))))
   (applicability-condition (kappa (?sol)
                                   (and (class ?sol)
                                        (exists ?sub (direct-subclass-of ?sub
?sol)))))))
;;;TASK REFINEMENT
(def-class refinement (goal-specification-task) ?task
  "The goal of this task is to take a solution  at a given
   level of refinement, say n, and to try refine it, using the
   given set of refiners. All refiners are tried for each solution at level n, but
   the refinement hiearchy is navigated down 1 level only.  That is, the output
will
   include at most solutions at level n+1.
   If a solution at level n cannot be refined, the method returns nil"
  ((has-input-role :value has-candidate-solution
                   :value has-observables
                   :value has-refiners)
   (has-output-role :value has-refined-solutions)
   (has-candidate-solution :type solution)
   (has-observables :type observables)
   (has-refined-solutions :type solution-space)
   (has-refiners :type list)
   (has-goal-expression
    :value
    (kappa (?task ?solutions)
           (forall ?sol
                   (=> (member ?sol ?solutions)
                       (exists ?ref
                               (and (member ?ref (role-value
                                                  ?task has-refiners))
                                    (member ?sol
                                            (apply-refiner-operator
                                             ?ref (role-value
                                                   ?task 'has-candidate-
solution))))))))))
```

```
;;;VANILLA-REFINEMENT-PSM
(def-class vanilla-refinement-psm (decomposition-method) ?psm
  "This method applies all refiners applicable to the given input solution"
  ((has-body
    :value
    '(lambda (?psm)
       (in-environment
        ((?sol . (role-value ?psm 'has-candidate-solution))
         (?refs . (achieve-generic-subtask
                   ?psm collect-refiners
                   has-solution ?sol
                   has-refiners (role-value ?psm has-refiners))))
        (achieve-generic-subtask ?psm apply-refiners
                                 has-solution ?sol
                                 has-refiners ?refs)))))
  :own-slots ((has-generic-subtasks '(collect-refiners
                                      apply-refiners ))
             (tackles-task-type 'refinement)))

;;;TASK COLLECT-REFINERS
(def-class  collect-refiners (goal-specification-task) ?task
  ((has-input-role :value has-solution
                   :value has-refiners)
   (has-output-role :value has-applicable-refiners)
   (has-solution :type solution)
   (has-refiners :type list)
   (has-applicable-refiners :type list)
   (has-goal-expression
    :value (kappa (?task ?refiners)
                  (and (subset ?refiners (role-value ?task has-refiners))
                       (not (exists (?ref)
                                    (and (member ?ref ?refiners)
                                         (not (refiner-is-applicable?
                                               ?ref (role-value
                                                     ?task has-solution))))))))))

;;;STANDARD-COLLECT-REFINERS-PSM
(def-class standard-collect-refiners-psm (primitive-method) ?psm
  ((has-body :value '(lambda (?psm)
                       (setofall ?x (and (member ?x (role-value ?psm has-refiners))
                                         (refiner-is-applicable?
                                          ?x (role-value ?psm has-solution)))))))

  :own-slots ((tackles-task-type 'collect-refiners)))
```

```
;;;TASK APPLY-REFINERS
(def-class  apply-refiners (goal-specification-task) ?task
  "The goal of this task is to generate all possible refinement of a solution,
given
   a set of refiners.  Only one-step refinement is carried out here.  That is, if
the
   input solution is defined at level n, the output can only be at level n+1."
  ((has-input-role :value has-solution
                   :value has-refiners)
   (has-output-role :value has-refined-solutions)
   (has-solution :type solution)
   (has-refiners :type list)
   (has-refined-solutions :type solution-space)
   (has-goal-expression
    :value (kappa (?task ?solutions)
                  (forall ?sol
                          (=> (member ?sol ?solutions)
                              (exists ?ref
                                      (and (member
                                            ?ref (role-value ?task 'has-refiners))
                                            (member ?sol
                                                    (apply-refiner-operator
                                                     ?ref
                                                     (role-value
                                                      ?task
                                                      'has-solution)))))))))))

;;;METHOD APPLY-REFINERS-PSM
(def-class apply-refiners-psm (problem-solving-method) ?psm
  ((has-control-role :value has-current-solutions)
   (has-current-solutions :type solution-space)
   (has-precondition :value (kappa (?psm)
                                   (every (role-value ?psm 'has-refiners)
                                          (kappa (?x)
                                                 (refiner-is-applicable?
                                                  ?x
                                                  (role-value
                                                   ?psm has-solution))))))

   (has-body :value
             '(lambda (?psm)
                (do
                  (set-role-value ?psm 'has-current-solutions nil)
                  (loop for ?ref in (role-value ?psm has-refiners)
                        do
                        (in-environment
                         ((?sols . (apply-refiner-operator
                                    ?ref
                                    (role-value ?psm 'has-solution))))
                         (set-role-value ?psm 'has-current-solutions
                                         (union (role-value ?psm
                                                           'has-current-solutions)
                                                ?sols))))
                  (role-value ?psm
                              'has-current-solutions)))))

  :own-slots ((tackles-task-type 'apply-refiners)
             ))
```