```
CK[Configuration: TYPE, FeatureExpression: TYPE,
    sat: [FeatureExpression, Configuration → boolean], FMi: TYPE, Feature: TYPE,
    [——]: [FMi → set[Configuration]], wf: [FMi → boolean],
    wt: [FMi, FeatureExpression → boolean],
    genFeatureExpression: [Feature → FeatureExpression], getFeatures: [FMi → set[Feature]],
    addMandatory: [FMi, FMi, Feature, Feature → bool],
    addOptional: [FMi, FMi, Feature, Feature → bool]]: THEORY
 BEGIN

  IMPORTING AssetMapping

  IMPORTING FMint[Configuration, FeatureExpression, sat]
                 {{FMi := FMi, Feature := Feature, [——] := [——], wf := wf, wt := wt,
                   getFeatures := getFeatures, addMandatory := addMandatory,
                   addOptional := addOptional}}

  RightSide: TYPE+

  Item: TYPE

  CK: TYPE

  am, am1, am2, pairs: VAR AM

  $a_1$, $a_2$, $a_3$: VAR Asset

  an, an1, an2: VAR AssetName

  anSet: VAR finite_sets[AssetName].finite_set

  aSet, $S_1$, $S_2$: VAR finite_sets[Asset].finite_set

  pair: VAR [Asset]

  fm, fm1, fm2: VAR FMi

  ck, ck1, ck2: VAR CK

  item, it, item1, item2: VAR Item

  its, its1, its2, commonIts, diffIts: VAR set[Item]
```

$c$, $c_1$, $c_2$: VAR Configuration

$e$, $e_1$, $e_2$: VAR FeatureExpression

$f$, $f_1$, $f_2$: VAR Feature

$p$, $p_1$, $p_2$: VAR Feature

$s$: VAR set[Configuration]

exps(ck): set[FeatureExpression]

items(ck): set[Item]

getExp(item): FeatureExpression

getRS(item): RightSide

wfCK(fm, am, ck): bool

semantics(ck)(am)($c$): finite_sets[Asset].finite_set

notshowupItem(it, an): bool

showupItem(it, an): bool

```
notshowupItems(it, anSet): bool =
    FORALL an: anSet(an) ⇒ notshowupItem(it, an)

notshowup(ck, an): bool =
    FORALL it: items(ck)(it) ⇒ notshowupItem(it, an)

showup(ck, an): bool =
    EXISTS it: items(ck)(it) ∧ showupItem(it, an)

itsFeature(its, f): bool =
    (FORALL c:
        FORALL it:
          its(it) ∧ sat(getExp(it), c) ⇒
            sat(genFeatureExpression(f), c))
```

$\diamond$: [FMi, CK, set[AssetName] $\rightarrow$ set[Configuration]]

```
filteredConfigurations: AXIOM
  FORALL s, fm, ck, anSet:
    (s ⊆ ◇ (fm, ck, anSet)) ⇒ (s ⊆ [——](fm))

falseExpMakesNoDiff: AXIOM
  FORALL (fm, am, ck1, ck2, s):
    (FORALL c:
        s(c) ⇒
         ((FORALL item: diffIts(item) ⇒ ¬ sat(getExp(item), c)) ⇒
            (semantics(ck1)(am)(c) = semantics(ck2)(am)(c))))
      WHERE diffIts = symmetric_difference(items(ck1), items(ck2))

syntaxChangeCKLine(ck1, ck2, item1, item2, its): bool =
    items(ck1) = (its ∪ {item1}) ∧
     items(ck2) = (its ∪ {item2})

syntaxReplaceFeatureExp(ck1, ck2, item1, item2, its): bool =
    syntaxChangeCKLine(ck1, ck2, item1, item2, its) ∧
     getRS(item1) = getRS(item2);

conditionsReplaceFeatureExp(fm, item1, item2): bool =
    wt(fm, getExp(item2)) ∧
     (FORALL c:
         [——](fm)(c) ⇒
          (sat(getExp(item1), c) ⇔ sat(getExp(item2), c)))

replaceFeatureExp_EqualCKeval: AXIOM
  FORALL (fm, am, ck1, ck2, item1, item2, its):
    ((wfCK(fm, am, ck1) ∧
         syntaxReplaceFeatureExp(ck1, ck2, item1, item2, its) ∧
          conditionsReplaceFeatureExp(fm, item1, item2))
       ⇒
       (FORALL c:
           [——](fm)(c) ⇒
            semantics(ck1)(am)(c) =
             semantics(ck2)(am)(c)))

syntaxSimpleDeleteAsset(ck, am1, am2, an): bool =
    dom(am1)(an) ∧ am2 = rm(an, am1) ∧ notshowup(ck, an);

simpleDeleteAsset_EqualCKeval: AXIOM
```

```
  FORALL (fm, ck, am1, am2, an):
    ((wfCK(fm, am1, ck) ∧ syntaxSimpleDeleteAsset(ck, am1, am2, an)) ⇒
      (FORALL c:
          [——](fm)(c) ⇒
           semantics(ck)(am1)(c) =
            semantics(ck)(am2)(c)))

syntaxAddMandatory(fm1, fm2, p, f): bool =
    getFeatures(fm1)(p) ∧
     ¬ (getFeatures(fm1)(f)) ∧ addMandatory(fm1, fm2, p, f);

addMandatory_EqualCKeval: AXIOM
  FORALL (fm1, fm2, am, ck, p, f):
    ((wfCK(fm1, am, ck) ∧ syntaxAddMandatory(fm1, fm2, p, f)) ⇒
      (FORALL c_1:
          [——](fm1)(c_1) ⇒
           (EXISTS c_2:
               [——](fm2)(c_2) ∧
                semantics(ck)(am)(c_1) =
                 semantics(ck)(am)(c_2))))

syntaxAddOptional(fm1, fm2, p, f, ck1, ck2, its, am1, am2, pairs): bool =
    getFeatures(fm1)(p) ∧
     ¬ (getFeatures(fm1)(f)) ∧
      addOptional(fm1, fm2, p, f) ∧
       items(ck2) = (items(ck1) ∪ its) ∧
        am2 = (am1 ∪ pairs) ∧
         (FORALL an: dom(pairs)(an) ⇒ ¬ (dom(am1)(an))) ∧
          itsFeature(its, f);

addOptional_EqualCKeval: AXIOM
  FORALL (fm1, fm2, am1, am2, ck1, ck2, p, f, its, pairs):
    ((wfCK(fm1, am1, ck1) ∧
         syntaxAddOptional(fm1, fm2, p, f, ck1, ck2, its, am1, am2, pairs))
      ⇒
      (FORALL c:
          [——](fm1)(c) ⇒
           [——](fm2)(c) ∧
            semantics(ck1)(am1)(c) =
             semantics(ck2)(am2)(c)))

syntaxChangeAsset(am1, am2, pairs, a_1, a_2, an): bool =
    am1 = ow((an, $a_1$), pairs) $\wedge$ am2 = ow((an, $a_2$), pairs)

sameEvalPairs: AXIOM
  FORALL (fm, am, ck, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset(am, am2, pairs, $a_1$, $a_2$, an) $\wedge$ $s$ = $\diamond$ (fm, ck, singleton(an)))
       $\Rightarrow$
       (FORALL $c$:
           $s(c)$ $\Rightarrow$
            (semantics(ck)(am)($c$)) =
             semantics(ck)(pairs)($c$)))

sameEvalPairs2: AXIOM
  FORALL (fm, am, ck, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset(am, am2, pairs, $a_1$, $a_2$, an) $\wedge$ $s$ = $\diamond$ (fm, ck, singleton(an)))
       $\Rightarrow$
       (FORALL $c$:
           $s(c)$ $\Rightarrow$
            (semantics(ck)(am2)($c$)) =
             semantics(ck)(pairs)($c$)))

syntaxAddAssets(am1, am2, ck1, ck2, pairs, its): bool =
    am2 = overw(pairs, am1) $\wedge$
     items(ck2) = (items(ck1) $\cup$ its)

conditionsAddAssets(pairs, its): bool =
    FORALL (item: Item):
      its(item) $\Rightarrow$
       (FORALL an: showupItem(item, an) $\Rightarrow$ dom(pairs)(an))

addAssetsSameProducts: AXIOM
  FORALL (fm, am, ck, am2, ck2, $s$, its, pairs):
    (($s$ = $\diamond$ (fm, ck2, domain(pairs)) $\wedge$
          syntaxAddAssets(am, am2, ck, ck2, pairs, its) $\wedge$ conditionsAddAssets(pairs, its))
       $\Rightarrow$
       (FORALL $c$:
           $s(c)$ $\Rightarrow$
            ((semantics(ck)(am)($c$)) =
                (semantics(ck2)(am2)($c$)))))

removeAssetsSameProducts: AXIOM
  FORALL (fm, am, ck, am2, ck2, $s$, its, pairs):
    (($s$ = $\diamond$ (fm, ck, domain(pairs)) $\wedge$
    syntaxAddAssets(am2, am, ck2, ck, pairs, its) $\wedge$ conditionsAddAssets(pairs, its))
  $\Rightarrow$
  (FORALL $c$:
      $s(c)$ $\Rightarrow$
        ((semantics(ck)(am)($c$)) =
            (semantics(ck2)(am2)($c$)))))

END CK
```

```
FMint[Configuration: TYPE, FeatureExpression: TYPE,
        sat: [FeatureExpression, Configuration → boolean]]: THEORY
 BEGIN

  IMPORTING FeatureExpression[Configuration]{{FeatureExpression := FeatureExpression, sat :=

  FMi: TYPE

  Feature: TYPE

  [——]: [FMi → set[Configuration]]

  ◇(fm: FMi, e: FeatureExpression): set[Configuration] =
      {c: Configuration | [——](fm)(c) ∧ ¬ sat(e, c)}

  ◇(e: FeatureExpression, fm: FMi): set[Configuration] =
      {c: Configuration | [——](fm)(c) ∧ sat(e, c)}

  ◇(fm: FMi, exps: set[FeatureExpression]): set[Configuration] =
      {c: Configuration |
           [——](fm)(c) ∧
            (FORALL (e: FeatureExpression):
                 exps(e) ⇒ ¬ sat(e, c))}

  wf(fm: FMi): boolean

  wt(fm: FMi, f: FeatureExpression): boolean

  genFeatureExpression(f: Feature): FeatureExpression

  getFeatures(fm: FMi): set[Feature]

  addMandatory(fm1: FMi, fm2: FMi, p: Feature, f: Feature): bool

  addOptional(fm1: FMi, fm2: FMi, p: Feature, f: Feature): bool

  addOR(fm1: FMi, fm2: FMi, p: Feature, f: Feature): bool

  addAlternative(fm1: FMi, fm2: FMi, p: Feature, f: Feature): bool

 END FMint
```

```
FeatureExpression[Configuration: TYPE]: THEORY
 BEGIN

  IMPORTING Configuration{{Configuration := Configuration}}

  FeatureExpression: TYPE

  sat(f: FeatureExpression, c: Configuration): boolean

 END FeatureExpression
```

```
Configuration: THEORY
 BEGIN

  Configuration: TYPE

 END Configuration
```

```
AssetMapping: THEORY
 BEGIN

  IMPORTING Assets, maps

  AM: TYPE = maps[AssetName, Asset].mapping

  am, am1, am2: VAR AM

  $a_1$, $a_2$, $a_3$: VAR Asset

  an, an1, an2: VAR AssetName

  anSet: VAR finite_sets[AssetName].finite_set

  aSet, $S_1$, $S_2$: VAR finite_sets[Asset].finite_set

  pair: VAR [Asset]

  pairs: VAR finite_sets[[Asset]].finite_set

  ▷(am1, am2): bool =
      (dom(am1) = dom(am2) ∧
          (FORALL an:
                dom(am1)(an) ⇒
                 (EXISTS $a_1$, $a_2$:
                       (am1(an, $a_1$)) ∧
                        (am2(an, $a_2$)) ∧
                         —-(singleton[Asset]($a_1$),
                              singleton[Asset]($a_2$)))))

  teste: THEOREM
    FORALL (am):
      dom(am)(an) ⇒
       (empty?(map(rm(an, am), singleton[AssetName](an))))

  testeNovo: THEOREM
    FORALL ($A$: AM):
      $A$ = (singleton[[Asset]](an, $a_1$) ∪ singleton[[Asset]](an, $a_2$)) ⇒
       unique((singleton[[Asset]](an, $a_1$) ∪ singleton  [[Asset]](an, $a_2$)))

  teste2: THEOREM
    FORALL (pairs):
      pairs = (singleton[[Asset]](an, $a_1$) ∪ singleton[[Asset]](an, $a_2$)) ∧
       ¬ ($a_1$ = $a_2$)
       ⇒ ¬ unique(pairs)

  assetMappingRefinement: THEOREM orders[AM].preorder?(▷)

  amRefCompositional: LEMMA
    FORALL (am1, am2):
      ▷(am1, am2) ⇒
       (FORALL (anSet):
          FORALL (aSet):
                wfProduct((aSet ∪ map(am1, anSet))) ⇒
                 wfProduct((aSet ∪ map(am2, anSet))) ∧
                  —-((aSet ∪ map(am1, anSet)),
                        (aSet ∪ map(am2, anSet))))

  renameAMitem(pair, an1, an2): [Asset] =
      IF (pair`1 = an1) THEN (an2, pair`2) ELSE pair ENDIF

  renameAM(pairs, an1, an2): set[[Asset]] =
      {$p$: [Asset] |
          EXISTS ($p_2$: [Asset]):
                pairs($p_2$) ∧ $p$ = renameAMitem($p_2$, an1, an2)}

  END AssetMapping
```

```
Assets: THEORY
 BEGIN

  IMPORTING set_aux_lemmas

  Asset: TYPE+

  AssetName: TYPE+

  CONVERSION+ singleton


  $a$, $a_1$, $a_2$, $a_3$: VAR Asset

  aSet, $S_1$, $S_2$: VAR set[Asset]

  —-: [set[Asset], set[Asset] → bool]

  wfProduct: [set[Asset] → bool]

  Product: TYPE = (wfProduct)

  assetRefinement: AXIOM orders[set[Asset]].preorder?(—-)

  asRefCompositional: AXIOM
    FORALL ($S_1$, $S_2$, aSet):
      ($S_1$ —- $S_2$) ∧ wfProduct(($S_1$ ∪ aSet)) ⇒
       wfProduct(($S_2$ ∪ aSet)) ∧
        ((($S_1$ ∪ aSet)) —- (($S_2$ ∪ aSet)))

  AssetTest: THEOREM
    FORALL ($S$, $T$, $x$, $y$: finite_sets[Asset].finite_set, $a$, $b$: Asset):
      wfProduct($x$) ∧
       ($S$ —- $T$) ∧
        $x$($a$) ∧
         (singleton[Asset]($a$) —- singleton[Asset]($b$)) ∧
          $x$ = ($S$ ∪ singleton[Asset]($a$)) ∧ $y$ = ($T$ ∪ singleton[Asset]($b$))
       ⇒ ($x$ —- $y$)

 END Assets
```

```
SPLPartialRefinement[Conf: TYPE, FM: TYPE, {——}: [FM → set[Conf]], Asset: TYPE,
                         AssetName: TYPE, CK: TYPE,
                         (IMPORTING maps[AssetName, Asset]) [——]: [CK →
                                                                  [mapping →
                                                                    [Conf →
                                                                      finite_sets
                                                                      [Asset].finite_
ORY
 BEGIN

  IMPORTING SPLPartialRefinementStrong[Conf, FM, {——}, Asset, AssetName, CK, [——]]

  IMPORTING SPLPartialRefinementWeak[Conf, FM, {——}, Asset, AssetName, CK, [——]]

  pl, pl1, pl2, pl3, pl4: VAR PL

  m: VAR CM

  p, p_1, p_2: VAR finite_sets[Asset].finite_set

  strongPartCaseWeak: THEOREM
    FORALL pl1, pl2, m:
      identity?(m) ⇒
        (strongPartialRefinement(pl1, pl2, domain(m)) ⇔
           weakPartialRefinement(pl1, pl2, m))

 END SPLPartialRefinement
```

```
SPLPartialRefinementStrong[Conf: TYPE, FM: TYPE, {——}: [FM → set[Conf]], As-
set: TYPE,
                                   AssetName: TYPE, CK: TYPE,
                                   (IMPORTING maps[AssetName, Asset]) [——]: [CK →
                                                                                   [mapping –
                                                                                     [Conf →
                                                                                       fi-
nite_sets
                                                                                       [Asset]
ORY
 BEGIN

  c, c_2: VAR Conf

  s, t: VAR set[Conf]

  fm, fm1, fm2: VAR FM

  IMPORTING SPLPartialRefinementCommon[Conf, FM, {——}, Asset, AssetName, CK, [——

  pl, pl1, pl2, pl3, pl4: VAR PL

  strongPartialRefinement(pl1, pl2, s): bool =
       (s ⊆ {——}(F(pl1))) ∧
        (s ⊆ {——}(F(pl2))) ∧
         (FORALL c: s(c) ⇒ (prod(pl1, c) —- prod(pl2, c)))

  strongPartRefReflexive: THEOREM
    FORALL pl, s: strongPartialRefinement(pl, pl, s)

  strongPartRefTransitive: THEOREM
    (FORALL pl1, pl2, pl3, s, t:
         (strongPartialRefinement(pl1, pl2, s) ∧ strongPartialRefinement(pl2, pl3, t)) ⇒
          strongPartialRefinement(pl1, pl3, (s ∩ t)))

  fmCompStrongDef: THEOREM
    FORALL (pl, fm1, fm2, s, t):
      ((s ⊆ t) ∧ fmPartialRefinement(fm1, fm2, t) ∧ wfPL(pl2) ⇒
          strongPartialRefinement(pl, pl2, s))
         WHERE fm1 = F(pl),
               pl2 = (#F := fm2, A := A(pl), K := K(pl)#)
```

```
fmCompStrongDefTest2: THEOREM
  FORALL (pl, fm2, s):
    (fmPartialRefinement(F(pl), fm2, s) ∧ wfPL(pl2) ⇒
       strongPartialRefinement(pl, pl2, s))
      WHERE fm1 = F(pl),
            pl2 = (#F := fm2, A := A(pl), K := K(pl)#)

partRefRel(pl1, pl2: PL, s: {confs: set[Conf] | (confs ⊆ {——}(F(pl1)))}):
      bool =
    FORALL c:
      s(c) ⇒
       (EXISTS c_2:
           {——}(F(pl2))(c_2) ∧
            (prod(pl1, c) —- prod(pl2, c_2)))

partRefFun(pl1, pl2: PL, s: set[Conf], f: [(s) → ({——}(F(pl2)))]): bool =
    FORALL (c: domain(f)):
      {——}(F(pl2))(f(c)) ∧
       (prod(pl1, c) —- prod(pl2, f(c)))

totalImpliesPartial: LEMMA
  FORALL pl1, pl2, (s: set[Conf] | (s ⊆ {——}(F(pl1)))):
    strongerPLrefinement(pl1, pl2) ⇒
     strongPartialRefinement(pl1, pl2, s)

partialImpliesTotal: LEMMA
  FORALL pl1, pl2, s:
    (s = {——}(F(pl1)) ∧ strongPartialRefinement(pl1, pl2, s)) ⇒
     strongerPLrefinement(pl1, pl2)

commutableDiagram: THEOREM
  FORALL pl1, pl3, pl4, (s: set[Conf] | (s ⊆ {——}(F(pl1)))):
    (strongerPLrefinement(pl1, pl3) ∧ strongPartialRefinement(pl3, pl4, s)) ⇒
     (EXISTS pl2:
         strongPartialRefinement(pl1, pl2, s) ∧
          strongerPLrefinement(pl2, pl4))

commutableDiagram2: THEOREM
  FORALL pl1, pl2, pl4, s:
    (strongPartialRefinement(pl1, pl2, s) ∧ strongerPLrefinement(pl2, pl4)) ⇒
     (EXISTS pl3:
         strongerPLrefinement(pl1, pl3) ∧
```

```
          strongPartialRefinement(pl3, pl4, s))

commutableDiagramAlt: THEOREM
  FORALL pl1, pl4, (s: set[Conf] | (s ⊆ {——}(F(pl1)))):
    (EXISTS pl2: strongPartialRefinement(pl1, pl2, s) ∧ strongerPLrefinement(pl2, pl4))
     ⇔
     (EXISTS pl3:
         strongerPLrefinement(pl1, pl3) ∧
          strongPartialRefinement(pl3, pl4, s))

partPlusTotalStrongerImpliesPart: THEOREM
  FORALL pl1, pl2, pl3, s:
    strongPartialRefinement(pl1, pl2, s) ∧ strongerPLrefinement(pl2, pl3) ⇒
     strongPartialRefinement(pl1, pl3, s)

totalStrongerPlusPartImpliesPart: THEOREM
  FORALL pl1, pl2, pl3, (s: set[Conf] | (s ⊆ {——}(F(pl1)))):
    strongerPLrefinement(pl1, pl2) ∧ strongPartialRefinement(pl2, pl3, s) ⇒
     strongPartialRefinement(pl1, pl3, s)

partPlusTotalImpliesPartRel: THEOREM
  FORALL pl1, pl2, pl3, s:
    strongPartialRefinement(pl1, pl2, s) ∧ plRefinement(pl2, pl3) ⇒
     partRefRel(pl1, pl3, s)

totalPlusPartImpliesPartRef: THEOREM
  FORALL pl1, pl2, pl3, s:
    plRefinement(pl1, pl2) ∧ strongPartialRefinement(pl2, pl3, s) ⇒
     (EXISTS (t: set[Conf]): partRefRel(pl1, pl3, t))

partRefExistsFunId: LEMMA
  FORALL pl1, pl2, s:
    strongPartialRefinement(pl1, pl2, s) ⇒
     (EXISTS (f: [(s) → (s)]):
         (FORALL c:
             s(c) ⇒
              ({——}(F(pl2))(f(c))) ∧
               (prod(pl1, c) —- prod(pl2, f(c)))))

partPlusTotalImpliesPartFun: THEOREM
  FORALL pl1, pl2, pl3, s:
    strongPartialRefinement(pl1, pl2, s) ∧ plRefinement(pl2, pl3) ⇒
```

```
(EXISTS (f: [(s) → ({——}(F(pl3)))]):
    (FORALL c:
        s(c) ⇒
         ({——}(F(pl3))(f(c)) ∧
            (prod(pl1, c) —- prod(pl3, f(c))))))

END SPLPartialRefinementStrong
```

```
SPLPartialRefinementWeak[Conf: TYPE, FM: TYPE, {——}: [FM → set[Conf]], As-
set: TYPE,
                              AssetName: TYPE, CK: TYPE,
                              (IMPORTING maps[AssetName, Asset]) [——]: [CK →
                                                                          [mapping →
                                                                            [Conf →
                                                                              fi-
nite_sets
                                                                               [Asset].fi
ORY
 BEGIN

  IMPORTING maps

  CM: TYPE = maps[Conf, Conf].mapping

  c: VAR Conf

  m, n: VAR CM

  IMPORTING maps_identity[Conf]

  IMPORTING maps_composite[Conf, Conf, Conf]

  IMPORTING SPLPartialRefinementCommon[Conf, FM, {——}, Asset, AssetName, CK, [——

  pl, pl1, pl2, pl3: VAR PL

  weakPartialRefinement(pl1, pl2, m): bool =
      (domain(m) ⊆ {——}(F(pl1))) ∧
       (image(m) ⊆ {——}(F(pl2))) ∧
        (FORALL c:
              domain(m)(c) ⇒
               (prod(pl1, c) —- prod(pl2, getRight(m, c))))

  weakPartRefReflexive: THEOREM
    FORALL pl, (m: CM | (domain(m) ⊆ {——}(F(pl)))):
      identity?(m) ⇒ weakPartialRefinement(pl, pl, m)

  weakPartRefTransitive: THEOREM
    FORALL pl1, pl2, pl3, m, n:
      ((weakPartialRefinement(pl1, pl2, m) ∧
        weakPartialRefinement(pl2, pl3, $n$) $\wedge$ image($m$) = domain($n$))
      $\Rightarrow$ weakPartialRefinement(pl1, pl3, $q$))
    WHERE $q$ = composeMaps($m$, $n$)

END SPLPartialRefinementWeak
```

```
maps_identity[S: TYPE]: THEORY
 BEGIN

  IMPORTING maps

  IMPORTING maps_composite[S, S, S]

  m: VAR maps[S, S].mapping

  identity?(m): bool = FORALL (l: S): getRight(m, l) = l

  composeIdResultsId: LEMMA
    FORALL m: identity?(m) ⇒ composeMaps(m, m) = m

  sameDomImg: LEMMA
    FORALL m: identity?(m) ⇒ domain(m) = image(m)

 END maps_identity
```

```
maps_composite[S: TYPE, T: TYPE, U: TYPE]: THEORY
 BEGIN

  IMPORTING maps

  m: VAR maps[S, T].mapping

  n: VAR maps[T, U].mapping

  l: VAR S

  r: VAR U

  composeMaps(m, n): maps[S, U].mapping =
      {l, r |
           domain(m)(l) ∧ r = getRight(n, getRight(m, l))}

  same_img: LEMMA
    FORALL m, n, l:
      (domain(q)(l) ⇒ getRight(q, l) = getRight(n, getRight(m, l)))
        WHERE q = composeMaps(m, n)

  domCompos: LEMMA
    FORALL m, n: (domain(composeMaps(m, n)) ⊆ domain(m))

  imgCompos: LEMMA
    FORALL m, n:
      image(m) = domain(n) ⇒
       image(composeMaps(m, n)) = image(n)

 END maps_composite
```

```
SPLPartialRefinementCommon[Conf: TYPE, FM: TYPE, {——}: [FM → set[Conf]], As-
set: TYPE,
                                       AssetName: TYPE, CK: TYPE,
                                       (IMPORTING maps[AssetName, Asset]) [——]: [CK →
                                                                                  [mapping –
                                                                                    [Conf →
                                                                                      fi-
nite_sets
                                                                                     [Asset]
ORY
 BEGIN

  IMPORTING maps

  AM: TYPE = maps[AssetName, Asset].mapping

  c: VAR Conf

  s: VAR set[Conf]

  fm1, fm2: VAR FM

  am, am1, am2: VAR AM

  an: VAR AssetName

  a_1, a_2: VAR Asset

  anSet: VAR set[AssetName]

  IMPORTING SPLrefinement[Conf, FM, Asset, AssetName, CK, {——}, [——]]

  fmPartialRefinement(fm1, fm2, s): bool =
      FORALL c: s(c) ⇒ {——}(fm1)(c) ∧ {——}(fm2)(c)

  fmPartRef: LEMMA
    FORALL fm1, fm2:
      (fm1 |= fm2) ⇔ fmPartialRefinement(fm1, fm2, {—fm1—})

  amPartialRefinement(am1, am2: AM,
                              anSet:
                                {aNames: set[AssetName] |
                                          (aNames ⊆ dom(am1)) ∧ (aNames ⊆ dom(am2))}):
      bool =
    (FORALL an:
        (anSet)(an) ⇒
         (EXISTS $a_1$, $a_2$:
              (am1(an, $a_1$)) ∧
               (am2(an, $a_2$)) ∧
                —-(singleton[Asset]($a_1$),
                      singleton[Asset]($a_2$))))

END SPLPartialRefinementCommon
```

```
SPLrefinement[Conf: TYPE, FM: TYPE, Asset: TYPE, AssetName: TYPE, CK: TYPE,
              {——}: [FM → set[Conf]],
              (IMPORTING maps[AssetName, Asset]) [——]: [CK →
                                                          [mapping →
                                                            [Conf →
                                                              finite_sets
                                                               [Asset].finite_set]]]]:
ORY
 BEGIN

  fm, fm1, fm2: VAR FM

  c, c_1, c_2, c_3: VAR Conf

  ⊨(fm1, fm2): bool = ({—fm1—} ⊆ {—fm2—})

  equivalentFMs(fm1, fm2): bool = {—fm1—} = {—fm2—}

  eqFM: THEOREM relations[FM].equivalence?(equivalentFMs)

  refFM: THEOREM orders[FM].preorder?(⊨)

  a_1, a_2: VAR Asset

  an, an1, an2: VAR AssetName

  aSet, S_1, S_2: VAR set[Asset]

  anSet: VAR finite_sets[AssetName].finite_set

  as1, as2, p, p_1, p_2: VAR finite_sets[Asset].finite_set

  prods, ps, ps1, ps2: VAR
          finite_sets[finite_sets[Asset].finite_set].finite_set

  —-: [set[Asset], set[Asset] → bool]

  wfProduct: [set[Asset] → bool]

  Product: TYPE = (wfProduct)

  assetRefinement: AXIOM orders[set[Asset]].preorder?(—-)
```

```
asRefCompositional: AXIOM
  FORALL ($S_1$, $S_2$, aSet):
    ($S_1$ —- $S_2$) ∧ wfProduct(($S_1$ ∪ aSet)) ⇒
     wfProduct(($S_2$ ∪ aSet)) ∧
      ((($S_1$ ∪ aSet)) —- (($S_2$ ∪ aSet)))

IMPORTING maps

AM: TYPE = maps[AssetName, Asset].mapping

am, am1, am2: VAR AM

▷(am1, am2): bool =
    (dom(am1) = dom(am2) ∧
       (FORALL an:
           dom(am1)(an) ⇒
            (EXISTS $a_1$, $a_2$:
                (am1(an, $a_1$)) ∧
                 (am2(an, $a_2$)) ∧
                  —-(singleton[Asset]($a_1$),
                      singleton[Asset]($a_2$)))))

assetMappingRefinement: THEOREM orders[AM].preorder?(▷)

amRefCompositional: LEMMA
  FORALL (am1, am2):
    ▷(am1, am2) ⇒
     (FORALL (anSet):
         FORALL (aSet):
           wfProduct((aSet ∪ map(am1, anSet))) ⇒
            wfProduct((aSet ∪ map(am2, anSet))) ∧
             —-((aSet ∪ map(am1, anSet)),
                 (aSet ∪ map(am2, anSet))))

amRef: AXIOM
  FORALL (am1, am2):
    ▷(am1, am2) ⇒
     (FORALL ($K$: CK, $c$: Conf):
         wfProduct([——]($K$)(am1)($c$)) ⇒
          wfProduct([——]($K$)(am2)($c$)) ∧
           ([——]($K$)(am1)($c$) —- [——]($K$)(am2)($c$)))
```

```
ck, ck1, ck2, ck3: VAR CK

equivalentCKs(ck1, ck2): bool = [—ck1—] = [—ck2—]

eqCK: THEOREM relations[CK].equivalence?(equivalentCKs)

weakerEqCK(fm, ck1, ck2): bool =
    FORALL am:
      FORALL c:
        {——}(fm)(c) ⇒
         [——](ck1)(am)(c) = [——](ck2)(am)(c)

weakerEqReflexive: THEOREM FORALL (fm, ck): weakerEqCK(fm, ck, ck)

weakerEqSymmetric: THEOREM
  FORALL (fm, ck1, ck2):
    weakerEqCK(fm, ck1, ck2) ⇒ weakerEqCK(fm, ck2, ck1)

weakerEqTransitive: THEOREM
  FORALL (fm, ck1, ck2, ck3):
    (weakerEqCK(fm, ck1, ck2) ∧ weakerEqCK(fm, ck2, ck3)) ⇒
     weakerEqCK(fm, ck1, ck3)

ArbitrarySPL: TYPE = [#F: FM, A: AM, K: CK#]

wfPL(pl: ArbitrarySPL): bool =
    (FORALL c:
        {——}(F(pl))(c) ⇒
         wfProduct([——](K(pl))(A(pl))(c)))

PL: TYPE = (wfPL)

pl, pl1, pl2: VAR PL

plRefinement(pl1, pl2): bool =
    (FORALL c_1:
        {——}(F(pl1))(c_1) ⇒
         (EXISTS c_2:
             {——}(F(pl2))(c_2) ∧
              (([——](K(pl1))(A(pl1))(c_1)) —-
                 ([——](K(pl2))(A(pl2))(c_2)))))
```

```
plRef: THEOREM orders[PL].preorder?(plRefinement)

products(pl): set[finite_sets[Asset].finite_set] =
    {p |
        EXISTS (c: Conf):
          ({——}(F(pl))(c)) ∧
           (p = ([——](K(pl))(A(pl))(c)))}

prod(pl, c): finite_sets[Asset].finite_set =
    ([——](K(pl))(A(pl))(c))

plRefinementAlt(pl1, pl2): bool =
    (FORALL $p_1$:
        products(pl1)($p_1$) ⇒
         (EXISTS $p_2$: products(pl2)($p_2$) ∧ (($p_1$) —- ($p_2$))))

plRefAlt: THEOREM orders[PL].preorder?(plRefinementAlt)

plRefEq: THEOREM
  FORALL (pl1, pl2):
    (plRefinement(pl1, pl2)) ⇔ (plRefinementAlt(pl1, pl2))

subsetProducts(prods, pl1): bool = (prods ⊆ products(pl1))

plWeakRefinement(pl1, pl2: PL, prods: {ps | (ps ⊆ products(pl1))}): bool =
    (FORALL $p_1$:
        prods($p_1$) ⇒
         (EXISTS $p_2$: products(pl2)($p_2$) ∧ (($p_1$) —- ($p_2$))))

strongerPLrefinement(pl1, pl2: PL): bool =
    (FORALL $c_1$:
        {——}(F(pl1))($c_1$) ⇒
         ({——}(F(pl2))($c_1$) ∧
            (([——](K(pl1))(A(pl1))($c_1$)) —-
               ([——](K(pl2))(A(pl2))($c_1$)))))

strongerPLref: THEOREM orders[PL].preorder?(strongerPLrefinement)

plStrongSubset: THEOREM
  FORALL (pl1, pl2):
    (strongerPLrefinement(pl1, pl2)) ⇒
    (({—F(pl1)—} ⊆ {—F(pl2)—}))

plRefinementFun(pl1, pl2: PL, f: [Conf → Conf]): bool =
  (FORALL c:
      {——}(F(pl1))(c) ⇒
       ({——}(F(pl2))(f(c)) ∧
          (prod(pl1, c) —- prod(pl2, f(c)))))

totalRefIFFExistsFun: LEMMA
 FORALL pl1, pl2:
   plRefinement(pl1, pl2) ⇔
    (EXISTS (f: [({——}(F(pl1))) → ({——}(F(pl2)))]):
        plRefinementFun(pl1, pl2, f))

weakFMcompositionality: THEOREM
 FORALL (pl, fm):
   ((F ⊨ fm) ∧ wfPL(pl2) ⇒ plRefinement(pl, pl2))
     WHERE F = F(pl),
           pl2 = (#F := fm, A := A(pl), K := K(pl)#)

fmEquivalenceCompositionality: THEOREM
 FORALL (pl, fm):
   (equivalentFMs(F, fm) ⇒ plRefinement(pl, pl2) ∧ wfPL(pl2))
     WHERE F = F(pl),
           pl2 = (#F := fm, A := A(pl), K := K(pl)#)

ckEquivalenceCompositionality: THEOREM
 FORALL (pl, ck):
   (equivalentCKs(K, ck) ⇒ plRefinement(pl, pl2) ∧ wfPL(pl2))
     WHERE K = K(pl),
           pl2 = (#F := F(pl), A := A(pl), K := ck#)

weakerCKcompositionality: THEOREM
 FORALL (pl, ck):
   (weakerEqCK(F, K, ck) ⇒ plRefinement(pl, pl2) ∧ wfPL(pl2))
     WHERE F = F(pl),
           K = K(pl),
           pl2 = (#F := F(pl), A := A(pl), K := ck#)

amRefinementCompositionality: THEOREM
 FORALL (pl, am):
   (⊳(A, am) ⇒ plRefinement(pl, pl2) ∧ wfPL(pl2))
```

```
      WHERE $A$ = $A$(pl),
            pl2 = (#$F$ := $F$(pl), $A$ := am, $K$ := $K$(pl)#)

fullCompositionality: THEOREM
  FORALL (pl, fm, am, ck):
    (equivalentFMs($F$, fm) $\wedge$ equivalentCKs($K$, ck) $\wedge$ $\rhd$($A$, am) $\Rightarrow$
      plRefinement(pl, pl2) $\wedge$ wfPL(pl2))
      WHERE $F$ = $F$(pl),
            $K$ = $K$(pl),
            $A$ = $A$(pl),
            pl2 = (#$F$ := fm, $A$ := am, $K$ := ck#)

weakFullCompositionality: THEOREM
  FORALL (pl, fm, am, ck):
    (($F$ $\models$ fm) $\wedge$ equivalentCKs($K$, ck) $\wedge$ $\rhd$($A$, am) $\wedge$ wfPL(pl2) $\Rightarrow$
      plRefinement(pl, pl2))
      WHERE $F$ = $F$(pl),
            $K$ = $K$(pl),
            $A$ = $A$(pl),
            pl2 = (#$F$ := fm, $A$ := am, $K$ := ck#)

fullCompositionality2: THEOREM
  FORALL (pl, fm, am, ck):
    (equivalentFMs($F$, fm) $\wedge$ weakerEqCK($F$, $K$, ck) $\wedge$ $\rhd$($A$, am) $\Rightarrow$
      plRefinement(pl, pl2) $\wedge$ wfPL(pl2))
      WHERE $F$ = $F$(pl),
            $K$ = $K$(pl),
            $A$ = $A$(pl),
            pl2 = (#$F$ := fm, $A$ := am, $K$ := ck#)

weakFullCompositionality2: THEOREM
  FORALL (pl, fm, am, ck):
    (($F$ $\models$ fm) $\wedge$ weakerEqCK($F$, $K$, ck) $\wedge$ $\rhd$($A$, am) $\wedge$ wfPL(pl2) $\Rightarrow$
      plRefinement(pl, pl2))
      WHERE $F$ = $F$(pl),
            $K$ = $K$(pl),
            $A$ = $A$(pl),
            pl2 = (#$F$ := fm, $A$ := am, $K$ := ck#)

singletonPL(pl): bool = singleton?(products(pl))

END SPLrefinement
```

```
maps[S: TYPE, T: TYPE]: THEORY
 BEGIN

  IMPORTING set_aux_lemmas

  $l, l_1, l_2$: VAR $S$

  $r, r_1, r_2$: VAR $T$

  $s$: VAR finite_sets[[T]].finite_set

  ls, ls1, ls2: VAR finite_sets[S].finite_set

  rs, rs1, rs2: VAR finite_sets[T].finite_set

  pair: VAR [T]

  unique($s$): bool =
      FORALL ($l, r_1, r_2$): ($s(l, r_1) \land s(l, r_2) \Rightarrow r_1 = r_2$)

  mapping: TYPE = (unique)

  $m, m_1, m_2$, pairs: VAR mapping

  pairs($m$): finite_sets[[T]].finite_set =
      {$p$: [T] | $m(p)$}

  dom($m$): set[S] = {$l$: $S$ | EXISTS ($r$: $T$): $m(l, r)$}

  domain($m$): finite_sets[S].finite_set =
      {$l$: $S$ | EXISTS ($r$: $T$): $m(l, r)$}

  indomain?($l, m$): bool = $\neg$ (dom($m$)($l$))

  img($m$): set[T] = {$r$: $T$ | EXISTS ($l$: $S$): $m(l, r)$}

  image($m$): finite_sets[T].finite_set =
      {$r$: $T$ | EXISTS ($l$: $S$): $m(l, r)$}

  mappingUnique: LEMMA
    FORALL ($m, l$):
      singleton?({$r$: $T$ | $m(l, r)$}) $\lor$
    empty?({r: T | m(l, r)})

inDom: LEMMA FORALL (m, l, r): m(l, r) ⇒ dom(m)(l)

map(m, ls): finite_sets[T].finite_set =
    {r: T | EXISTS (l: S): ls(l) ∧ m(l, r)}

getRight(m: mapping, l: {n: S | dom(m)(n)}): T =
    singleton_elt[T]
        (map(m,
              extend[S, {n: S | dom(m)(n)}, bool, FALSE]
                  (singleton(l))))

unmap(m, rs): finite_sets[S].finite_set =
    {l: S | EXISTS (r: T): rs(r) ∧ m(l, r)}

uniqueUnion: LEMMA
  FORALL (m_1, m_2):
    (FORALL l: dom(m_2)(l) ⇒ ¬ (dom(m_1)(l))) ⇒
     unique((m_1 ∪ m_2))

uniqueSingleton: LEMMA
  FORALL (pair): unique(singleton[[T]](pair))

domUnion: LEMMA
  FORALL (m_1, m_2):
    (FORALL l: dom(m_2)(l) ⇒ ¬ (dom(m_1)(l))) ⇒
     dom((m_1 ∪ m_2)) = (dom(m_1) ∪ dom(m_2))

unionMap: LEMMA
  FORALL (m, ls1, ls2):
    map(m, (ls1 ∪ ls2)) = (map(m, ls1) ∪ map(m, ls2))

existsMap: LEMMA
  FORALL (m, l, r):
    m(l, r) ⇒ map(m, singleton(l)) = singleton(r)

rm(l, m): mapping =
    IF (dom(m)(l))
      THEN (m \ {(λ (x : [finite_sets[T].finite_set]) : (x‘1, singleton_elt [T](x‘2)))(l, map(m,
    ELSE m
    ENDIF
```

```
remove(ls, $m$): mapping =
    {pair | $m$(pair) $\wedge$ ($\neg$ ls(pair`1))}

filter(ls, $m$): mapping = {pair | $m$(pair) $\wedge$ ls(pair`1)}

ow(pair, $m$): mapping =
    IF (dom($m$)(pair`1))
      THEN (singleton[[$T$]](pair) $\cup$ rm(pair`1, $m$))
    ELSE (singleton[[$T$]](pair) $\cup$ $m$)
    ENDIF

overw(pairs, $m$): mapping =
    (pairs $\cup$ remove(domain(pairs), $m$))

uniqueUnionRM: LEMMA
  FORALL ($m$, $l$, $r$):
    unique((singleton[[$T$]]($l$, $r$) $\cup$ rm($l$, $m$)))

domainContained: LEMMA
  FORALL ($m$, $l$, $r$): (dom($m$) $\subseteq$ dom(ow(($l$, $r$), $m$)))

mapOR: LEMMA
  FORALL ($m$, $m_1$, $l$, $r$):
    $m$ = ow(($l$, $r$), $m_1$) $\Rightarrow$
     map($m$, singleton($l$)) = singleton($r$)

mapUnion: LEMMA
  FORALL ($m$, ls1, ls2, $r$):
    map($m$, (ls1 $\cup$ ls2))($r$) $\Rightarrow$
     map($m$, ls1)($r$) $\vee$ map($m$, ls2)($r$)

mapAM: LEMMA
  FORALL ($m$, $l$, ls):
    dom($m$)($l$) $\Rightarrow$
     (EXISTS ($r$: $T$):
         $m$($l$, $r$) $\wedge$
          map($m$, (singleton[$S$]($l$) $\cup$ ls)) =
           (singleton[$T$]($r$) $\cup$ map($m$, ls)))

notExists: LEMMA
  FORALL ($m$, ls):
        ¬ (EXISTS (l: S): ls(l) ∧ dom(m)(l)) ⇒
         map(m, ls) = ∅

  getRightResult: LEMMA
    FORALL m, r, (l: {n: S | dom(m)(n)}):
      m(l, r) ⇒ getRight(m, l) = r

END maps
```

```
set_aux_lemmas[T: TYPE]: THEORY
 BEGIN

  CONVERSION+ singleton


  cardUnion: LEMMA
    FORALL (an: T, anSet: finite_sets[T].finite_set):
      (¬ (an ∈ anSet)) ⇒
       finite_sets[T].Card(anSet) <
        finite_sets[T].Card((singleton[T](an) ∪ anSet))

  setMember: LEMMA
    FORALL (anSet: finite_sets[T].finite_set, an: T):
      (an ∈ anSet) ⇒
       (EXISTS (anSet2: finite_sets[T].finite_set):
           anSet = (singleton[T](an) ∪ anSet2) ∧
            (¬ (an ∈ anSet2)))

  lemmaUnionRemove: LEMMA
    FORALL (X, Y: finite_sets[T].finite_set, item: T):
      Y = (X \ {item}) ∧ (item ∈ X) ⇒
       X = (Y ∪ singleton[T](item))

  lemmaUnionRemove2: LEMMA
    FORALL (X: finite_sets[T].finite_set, item: T):
      (item ∈ X) ⇒
       (singleton[T](item) ∪ (X \ {item})) = X

  lemmaUnionRemove3: LEMMA
    FORALL (X: finite_sets[T].finite_set, item1, item2: T):
      (item1 ∈ X) ∧ (item2 ∈ X) ⇒
       ((singleton[T](item1) ∪ singleton[T](item2)) ∪ ((X \ {item2}) \ {item1})) =
        X

  finiteIntersection: LEMMA
    FORALL (A: finite_sets[T].finite_set, B: set[T]):
      is_finite[T]((A ∩ B))

  finiteComprehension: LEMMA
    FORALL (S: finite_sets[T].finite_set):
      is_finite({x: T | S(x)})
```

finiteUnion: LEMMA
  FORALL $(X,\ Y\colon \text{set}[T])$:
    $(\text{is\_finite}[T](X)\ \wedge\ \text{is\_finite}[T](Y))\ \Rightarrow$
     $\text{is\_finite}[T]((X \cup Y))$

singletonMember: LEMMA
  FORALL $(x,\ y\colon T)$: $\text{singleton}(x)(y)\ \Rightarrow\ x\ =\ y$

singletonEqualMember: LEMMA
  FORALL $(x,\ y\colon T,\ S\colon \text{set}[T])$:
    $\text{singleton?}(S)\ \wedge\ S(x)\ \wedge\ S(y)\ \Rightarrow\ x\ =\ y$

memberUnion: LEMMA
  FORALL $(x,\ y\colon T,\ S\colon \text{set}[T])$:
    $S(x)\ \Rightarrow\ (x \in (S \cup \text{singleton}[T](y)))$

intersectionNotMember: LEMMA
  FORALL $(x,\ y\colon \text{finite\_sets}[T].\text{finite\_set},\ e\colon T)$:
    $x(e)\ \wedge\ (x \cap y)\ =\ \emptyset\ \Rightarrow\ \neg\ y(e)$

intersectionSubset: LEMMA
  FORALL $(x,\ y,\ z\colon \text{finite\_sets}[T].\text{finite\_set})$:
    $(x \cap y)\ =\ \emptyset\ \wedge\ (z \subseteq y)\ \Rightarrow\ (x \cap z)\ =\ \emptyset$

disjointUnion: LEMMA
  FORALL $(x,\ y\colon \text{finite\_sets}[T].\text{finite\_set},\ e\colon T)$:
    $y(e)\ \wedge\ \text{disjoint?}(x,\ y)\ \Rightarrow\ \neg\ (x(e))$

disjointSubset: LEMMA
  FORALL $(x,\ y,\ z\colon \text{finite\_sets}[T].\text{finite\_set},\ e\colon T)$:
    $y(e)\ \wedge\ \text{disjoint?}(x,\ y)\ \wedge\ y\ =\ (\text{singleton}[T](e) \cup z)\ \wedge\ \neg\ (z(e))$
     $\Rightarrow\ \text{disjoint?}(x,\ z)$

unionRemoveEqual: LEMMA
  FORALL $(x\colon \text{finite\_sets}[T].\text{finite\_set},\ m,\ n\colon T)$:
    $x\ =\ (\text{singleton}[T](m) \cup (x \setminus \{n\}))\ \wedge\ x(n)\ \Rightarrow$
     $m\ =\ n$

unionRemoveEqual2: LEMMA
  FORALL $(x,\ y\colon \text{finite\_sets}[T].\text{finite\_set},\ m\colon T)$:
    $\text{union}(x,\ y)(m)\ \Rightarrow$

```
      (x ∪ y) =
        ((x ∪ singleton[T](m)) ∪ ((x ∪ y) \ {m}))

  singletonEqual: LEMMA
    FORALL (m: T):
      singleton[T](m) =
        extend[T, {a: T | singleton[T](m)(a)}, bool, FALSE]
              (singleton[{a: T | singleton[T](m)(a)}](m))
        ∧
        singleton[{a: T | singleton[T](m)(a)}](m) =
          restrict[T, {a: T | singleton[T](m)(a)}, boolean]
                (singleton[T](m))

END set_aux_lemmas
```

```
SPLStrongPartRefTemplInt[Configuration: TYPE, FeatureExpression: TYPE,
                         sat: [FeatureExpression, Configuration → boolean], FMi: TYPE,
                         Feature: TYPE, [——]: [FMi → set[Configuration]],
                         wf: [FMi → boolean], wt: [FMi, FeatureExpression → boolean],
                         genFeatureExpression: [Feature → FeatureExpression],
                         getFeatures: [FMi → set[Feature]],
                         addMandatory: [FMi, FMi, Feature, Feature → bool],
                         addOptional: [FMi, FMi, Feature, Feature → bool]]: THE-
ORY
 BEGIN

  IMPORTING CK
              [Configuration, FeatureExpression, sat, FMi, Feature, [——], wf, wt,
                genFeatureExpression, getFeatures, addMandatory, addOptional]

  IMPORTING AssetMapping

  fm: VAR FMi

  am, am2, pairs: VAR AM

  $a_1$, $a_2$: VAR Asset

  an: VAR AssetName

  ck1, ck2: VAR CK

  item, item1, item2: VAR Item

  its: VAR set[Item]

  $c$: VAR Configuration

  $s$: VAR set[Configuration]

  exp: VAR FeatureExpression

  IMPORTING SPLPartialRefinement[Configuration, FMi, [——], Asset, AssetName, CK, se-
mantics]

  pl, pl2: VAR PL
```

```
changeCKLineStrongPartialRef: THEOREM
  FORALL (pl, ck2, item1, item2, its, $s$):
    ((wfCK($F$(pl), $A$(pl), $K$(pl)) $\wedge$
          $s$ = ($\diamond$ ($F$(pl), getExp(item1)) $\cap$ $\diamond$ ($F$(pl), getExp(item2))) $\wedge$
           syntaxChangeCKLine($K$(pl), $K$(pl2), item1, item2, its) $\wedge$
            wt($F$(pl), getExp(item2)))
        $\Rightarrow$ strongPartialRefinement(pl, pl2, $s$))
       WHERE pl2 = (#$F$ := $F$(pl), $A$ := $A$(pl), $K$ := ck2#)

addCKLinesStrongPartialRef: THEOREM
  FORALL (pl, ck2, its, $s$):
    ((wfCK($F$(pl), $A$(pl), $K$(pl)) $\wedge$
          $s$ = $\diamond$ ($F$(pl), {exp | EXISTS item: its(item) $\wedge$ exp = getExp(item)}) $\wedge$
           items(ck2) = (its $\cup$ items($K$(pl))))
        $\Rightarrow$ strongPartialRefinement(pl, pl2, $s$))
       WHERE pl2 = (#$F$ := $F$(pl), $A$ := $A$(pl), $K$ := ck2#)

removeCKLinesStrongPartialRef: THEOREM
  FORALL (pl, ck2, its, $s$):
    ((wfCK($F$(pl), $A$(pl), $K$(pl)) $\wedge$
          $s$ = $\diamond$ ($F$(pl), {exp | EXISTS item: its(item) $\wedge$ exp = getExp(item)}) $\wedge$
           items($K$(pl)) = (its $\cup$ items(ck2)))
        $\Rightarrow$ strongPartialRefinement(pl, pl2, $s$))
       WHERE pl2 = (#$F$ := $F$(pl), $A$ := $A$(pl), $K$ := ck2#)

changeAssetStrongPartialRef: THEOREM
  FORALL (pl, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset($A$(pl), am2, pairs, $a_1$, $a_2$, an) $\wedge$
          $s$ = $\diamond$ ($F$(pl), $K$(pl), singleton(an)))
        $\Rightarrow$ strongPartialRefinement(pl, pl2, $s$))
       WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := $K$(pl)#)

addAssetsStrongPartialRef: THEOREM
  FORALL (pl, am2, ck2, $s$, its, pairs):
    (($s$ = $\diamond$ ($F$(pl2), $K$(pl2), domain(pairs)) $\wedge$
          syntaxAddAssets($A$(pl), am2, $K$(pl), ck2, pairs, its) $\wedge$
           conditionsAddAssets(pairs, its) $\wedge$
            (FORALL $c$:
                 $\neg$ $s$($c$) $\Rightarrow$
                  SPLrefinement.wfProduct(semantics($K$(pl2))($A$(pl2))($c$))))
        $\Rightarrow$ strongPartialRefinement(pl, pl2, $s$))
       WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := ck2#)
```

```
removeAssetsStrongPartialRef: THEOREM
  FORALL (pl, am2, ck2, $s$, its, pairs):
    (($s$ = ◇ ($F$(pl), $K$(pl), domain(pairs)) ∧
          syntaxAddAssets(am2, $A$(pl), ck2, $K$(pl), pairs, its) ∧
           conditionsAddAssets(pairs, its) ∧
            (FORALL $c$:
                 ¬ $s(c)$ ⇒
                  SPLrefinement.wfProduct(semantics($K$(pl2))($A$(pl2))($c$))))
       ⇒ strongPartialRefinement(pl, pl2, $s$))
      WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := ck2#)

END SPLStrongPartRefTemplInt
```

```
SPLPartialRefTemplates: THEORY
 BEGIN

  IMPORTING FeatureModel, Name, FeatureModelSemantics, FeatureModelRefinements

  IMPORTING Assets, AssetMapping, ConfigurationKnowledge

  aSet: VAR finite_sets[Asset].finite_set

  am1, am2, pairs: VAR AM

  a_1, a_2: VAR Asset

  an: VAR AssetName

  anSet: VAR finite_sets[AssetName].finite_set

  s, t: VAR set[Configuration]

  c: VAR Configuration

  fm, fm1, fm2: VAR FM

  ck, ck1, ck2, its: VAR CK

  item1, item2: VAR Item

  items: VAR set[Item]

  P, Q: VAR Name

  exp: VAR Formula_

  IMPORTING SPLPartialRefinement
                [Configuration, WFM, restrict[FM, WFM, set[Configuration]](semantics),
                  Assets.Asset, Assets.AssetName, CK, semantics]

  pl, pl2, pl3: VAR PL

  m: VAR CM

  ◇(fm, ck, anSet): set[Configuration] =
    {$c$ |
        semantics(fm)($c$) ∧
         (FORALL ($i$: Item):
             evalCK(ck, $c$)($i$) ⇒
              empty?((getRS($i$) ∩ anSet)))}

◇(fm, exp): set[Configuration] =
    {$c$ | semantics(fm)($c$) ∧ satisfies(exp, $c$)}

syntaxChangeAsset(am1, am2, pairs, $a_1$, $a_2$, an): bool =
    am1 = ow((an, $a_1$), pairs) ∧ am2 = ow((an, $a_2$), pairs)

sameEvalPairs: LEMMA
  FORALL (pl, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset($A$(pl), am2, pairs, $a_1$, $a_2$, an) ∧
         $s$ = ◇ ($F$(pl), $K$(pl), singleton(an)))
       ⇒
       (FORALL $c$:
           $s$($c$) ⇒
            (semantics($K$(pl))($A$(pl))($c$)) =
             semantics($K$(pl2))(pairs)($c$)))
      WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := $K$(pl)#)

sameEvalPairs2: LEMMA
  FORALL (pl, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset($A$(pl), am2, pairs, $a_1$, $a_2$, an) ∧
         $s$ = ◇ ($F$(pl), $K$(pl), singleton(an)))
       ⇒
       (FORALL $c$:
           $s$($c$) ⇒
            (semantics($K$(pl))(am2)($c$)) = semantics($K$(pl2))(pairs)($c$)))
      WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := $K$(pl)#)

changeAssetSameProducts: THEOREM
  FORALL (pl, am2, pairs, $a_1$, $a_2$, an, $s$):
    ((syntaxChangeAsset($A$(pl), am2, pairs, $a_1$, $a_2$, an) ∧
         $s$ = ◇ ($F$(pl), $K$(pl), singleton(an)))
       ⇒
       (FORALL $c$:
           $s$($c$) ⇒
            ((semantics($K$(pl))($A$(pl))($c$)) =
                (semantics($K$(pl2))(am2)($c$)))))
        WHERE pl2 = (#F := F(pl), A := am2, K := K(pl)#)

changeAssetStrong: THEOREM
  FORALL (pl, am2, pairs, a_1, a_2, an, s):
    ((syntaxChangeAsset(A(pl), am2, pairs, a_1, a_2, an) ∧
         s = ◇ (F(pl), K(pl), singleton(an)) ∧
          (FORALL c:
              ¬ s(c) ⇒
               SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
      ⇒ strongPartialRefinement(pl, pl2, s))
        WHERE pl2 = (#F := F(pl), A := am2, K := K(pl)#)

changeAssetWeak: THEOREM
  FORALL (pl, am2, pairs, a_1, a_2, an, m):
    ((syntaxChangeAsset(A(pl), am2, pairs, a_1, a_2, an) ∧
         domain(m) = ◇ (F(pl), K(pl), singleton(an)) ∧
          identity?(m) ∧
           (FORALL c:
               ¬ domain(m)(c) ⇒
                SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
      ⇒ weakPartialRefinement(pl, pl2, m))
        WHERE pl2 = (#F := F(pl), A := am2, K := K(pl)#)

transfOptMand(fm1, fm2, P, Q): bool =
    features(fm1) = features(fm2) ∧
     formulae(fm2) =
      (formulae(fm1) ∪ singleton [(IMPLIES?)](IMPLIES_FORMULA(NAME_FORMULA(P), N

syntaxTransfOptMand(fm1, fm2, P, Q): bool =
    transfOptMand(fm1, fm2, P, Q) ∧
     (features(fm1))(P) ∧ (features(fm1))(Q)

conditionsTransfOptMand(fm1, P, Q): bool =
    FORALL c:
      semantics(fm1)(c) ⇒
       satisfies(IMPLIES_FORMULA(NAME_FORMULA(Q), NAME_FORMULA(P)),
                 c)

wfTransfOptMand: THEOREM
  FORALL (pl, fm2, P, Q):
    ((syntaxTransfOptMand(F(pl), fm2, P, Q) ∧
         conditionsTransfOptMand(F(pl), P, Q))
```

```
          ⇒ wfFM(fm2) ∧ wfPL(pl2))
        WHERE pl2 = (#F := fm2, A := A(pl), K := K(pl)#)

transOptMandPartRefStrong: THEOREM
  FORALL (pl, fm2, s, P, Q):
    ((syntaxTransfOptMand(F(pl), fm2, P, Q) ∧
          conditionsTransfOptMand(F(pl), P, Q) ∧
           s = ◇ (F(pl), (IMPLIES_FORMULA(NAME_FORMULA(P), NAME_FORMULA(
          ⇒ strongPartialRefinement(pl, pl2, s))
        WHERE pl2 = (#F := fm2, A := A(pl), K := K(pl)#)

transOptMandPartRefWeak: THEOREM
  FORALL (pl, fm2, m, P, Q):
    ((syntaxTransfOptMand(F(pl), fm2, P, Q) ∧
          conditionsTransfOptMand(F(pl), P, Q) ∧
           domain(m) =
            ◇ (F(pl), (IMPLIES_FORMULA(NAME_FORMULA(P), NAME_FORMULA(Q)))
            ∧ identity?(m))
          ⇒ weakPartialRefinement(pl, pl2, m))
        WHERE pl2 = (#F := fm2, A := A(pl), K := K(pl)#)

syntaxChangeCKLine(ck1, ck2, item1, item2, items): bool =
    ck1 = (singleton[Item](item1) ∪ items) ∧
     ck2 = (singleton[Item](item2) ∪ items)

conditionsChangeCKLine(fm, item1, item2): bool =
    wt(fm, exp(item2))

predChangeCKLine(pl, ck2, item1, item2, items, s): bool =
    (syntaxChangeCKLine(K(pl), ck2, item1, item2, items) ∧
        conditionsChangeCKLine(F(pl), item1, item2) ∧
         s =
          ◇ (F(pl),
               AND_FORMULA(NOT_FORMULA(exp(item1)),
                           NOT_FORMULA(exp(item2)))))

changeCKLineSameEvalCK: LEMMA
  FORALL (pl, ck2, item1, item2, items, s):
    ((predChangeCKLine(pl, ck2, item1, item2, items, s) ∧
         (FORALL c:
               ¬ s(c) ⇒
                SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
        ⇒
        (FORALL c:
             s(c) ⇒
              ((semantics(K(pl))(A(pl))(c)) =
                 (semantics(K(pl2))(A(pl2))(c)))))
       WHERE pl2 = (#F := F(pl), A := A(pl), K := ck2#)

changeCKLineStrongPartRef: THEOREM
  FORALL (pl, ck2, item1, item2, items, s):
    ((predChangeCKLine(pl, ck2, item1, item2, items, s) ∧
         (FORALL c:
              ¬ s(c) ⇒
               SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
        ⇒ strongPartialRefinement(pl, pl2, s))
       WHERE pl2 = (#F := F(pl), A := A(pl), K := ck2#)

changeCKLineWeakPartRef: THEOREM
  FORALL (pl, ck2, item1, item2, items, m):
    ((predChangeCKLine(pl, ck2, item1, item2, items, domain(m)) ∧
         (FORALL c:
              ¬ domain(m)(c) ⇒
               SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c)))
          ∧ identity?(m))
        ⇒ weakPartialRefinement(pl, pl2, m))
       WHERE pl2 = (#F := F(pl), A := A(pl), K := ck2#)

filterFormulae(fm, Q): set[Formula_] =
    {form: Formula_ |
        formulae(fm)(form) ∧ ¬ (Q ∈ names(form))}

removeFeature(fm1, fm2, P, Q): bool =
    formulae(fm2) = filterFormulae(fm1, Q) ∧
     features(fm2) = (features(fm1) \ {Q})

syntaxRemoveFeature(fm1, fm2, am1, am2, ck1, ck2, P, Q, its, pairs): bool =
    removeFeature(fm1, fm2, P, Q) ∧
     features(fm1)(P) ∧
      features(fm1)(Q) ∧
       am1 = overw(pairs, am2) ∧ ck2 = (ck1 \ its)

conditionsOpt(fm1, P, Q): bool =
    FORALL c:
      semantics(fm1)($c$) $\Rightarrow$
       $\neg$ satisfies(IMPLIES_FORMULA(NAME_FORMULA($P$), NAME_FORMULA($Q$)),
                   $c$)

conditionsMand(fm1, $P$, $Q$): bool =
    FORALL $c$:
      semantics(fm1)($c$) $\Rightarrow$
       satisfies(IMPLIES_FORMULA(NAME_FORMULA($P$), NAME_FORMULA($Q$)),
                 $c$)

conditionsRemoveFeature(fm1, its, pairs, $P$, $Q$, ck): bool =
    (FORALL $c$:
        FORALL exp:
          exps(ck)(exp) $\wedge$ satisfies(exp, $c$) $\Rightarrow$
           (exps(its)(exp) $\Leftrightarrow$ satisfies(NAME_FORMULA($Q$), $c$)))
     $\wedge$
     (FORALL (item: Item):
         $\neg$ its(item) $\Rightarrow$ (FORALL an: (assets(item))(an) $\Rightarrow$ $\neg$ dom(pairs)(an)))
      $\wedge$
      (FORALL $c$:
          semantics(fm1)($c$) $\Rightarrow$
           satisfies(IMPLIES_FORMULA(NAME_FORMULA($Q$), NAME_FORMULA($P$)), $c$))
       $\wedge$
       (conditionsOpt(fm1, $P$, $Q$) $\vee$ conditionsMand(fm1, $P$, $Q$))

predRemoveFeature(pl, pl2, $s$, its, pairs, $P$, $Q$): bool =
    (syntaxRemoveFeature($F$(pl), $F$(pl2), $A$(pl), $A$(pl2), $K$(pl), $K$(pl2), $P$, $Q$,
                         its, pairs)
       $\wedge$
       conditionsRemoveFeature($F$(pl), its, pairs, $P$, $Q$, $K$(pl)) $\wedge$
        $s$ = $\Diamond$ ($F$(pl), NOT_FORMULA(NAME_FORMULA($Q$))))

itsNotIncluded: LEMMA
  FORALL (pl, pl2, $s$, its, pairs, $P$, $Q$):
    (predRemoveFeature(pl, pl2, $s$, its, pairs, $P$, $Q$) $\Rightarrow$
       (FORALL $c$:
           $s$($c$) $\Rightarrow$
            (FORALL ($i$: Item):
                evalCK($K$(pl), $c$)($i$) $\Rightarrow$ $\neg$ its($i$))))

pairsNotIncluded: LEMMA
  FORALL (pl, pl2, $s$, its, pairs, $P$, $Q$):
    (predRemoveFeature(pl, pl2, $s$, its, pairs, $P$, $Q$) $\Rightarrow$
        (FORALL $c$:
            $s(c)$ $\Rightarrow$
             (FORALL an:
                  eval($K$(pl), $c$)(an) $\Rightarrow$ $\neg$ dom(pairs)(an))))

removeFeatureSameProducts: THEOREM
  FORALL (pl, pl2, $s$, its, pairs, $P$, $Q$):
    (predRemoveFeature(pl, pl2, $s$, its, pairs, $P$, $Q$) $\Rightarrow$
        (FORALL $c$: $s(c)$ $\Rightarrow$ prod(pl, $c$) = prod(pl2, $c$)))

removeFeaturePartRefStrong: THEOREM
  FORALL (pl, pl2, $s$, its, pairs, $P$, $Q$):
    (predRemoveFeature(pl, pl2, $s$, its, pairs, $P$, $Q$) $\Rightarrow$
        strongPartialRefinement(pl, pl2, $s$))

removeFeaturePartRefWeak: THEOREM
  FORALL (pl, pl2, $m$, its, pairs, $P$, $Q$):
    ((identity?($m$) $\wedge$ predRemoveFeature(pl, pl2, domain($m$), its, pairs, $P$, $Q$)) $\Rightarrow$
        weakPartialRefinement(pl, pl2, $m$))

syntaxAddAssets(am1, am2, ck1, ck2, pairs, its): bool =
    am2 = overw(pairs, am1) $\wedge$ ck2 = (ck1 $\cup$ its)

conditionsAddAssets(pairs, its): bool =
    FORALL (item: Item):
      its(item) $\Rightarrow$ (assets(item) $\subseteq$ dom(pairs))

addAssetsSameProducts: THEOREM
  FORALL (pl, am2, ck2, $s$, its, pairs):
    (($s$ = $\diamond$ ($F$(pl2), $K$(pl2), domain(pairs)) $\wedge$
          syntaxAddAssets($A$(pl), am2, $K$(pl), ck2, pairs, its) $\wedge$
           conditionsAddAssets(pairs, its))
        $\Rightarrow$
        (FORALL $c$:
            $s(c)$ $\Rightarrow$
             ((semantics($K$(pl))($A$(pl))($c$)) =
                 (semantics($K$(pl2))($A$(pl2))($c$)))))
      WHERE pl2 = (#$F$ := $F$(pl), $A$ := am2, $K$ := ck2#)

addAssetsPartRefStrong: THEOREM
  FORALL (pl, am2, ck2, $s$, its, pairs):
    ((s = ◇ (F(pl2), K(pl2), domain(pairs)) ∧
          syntaxAddAssets(A(pl), am2, K(pl), ck2, pairs, its) ∧
           conditionsAddAssets(pairs, its) ∧
            (FORALL c:
                 ¬ s(c) ⇒
                  SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
       ⇒ strongPartialRefinement(pl, pl2, s))
      WHERE pl2 = (#F := F(pl), A := am2, K := ck2#)

addAssetsPartRefWeak: THEOREM
  FORALL (pl, am2, ck2, m, its, pairs):
    ((domain(m) = ◇ (F(pl2), K(pl2), domain(pairs)) ∧
          syntaxAddAssets(A(pl), am2, K(pl), ck2, pairs, its) ∧
           conditionsAddAssets(pairs, its) ∧
            identity?(m) ∧
             (FORALL c:
                  ¬ domain(m)(c) ⇒
                   SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
       ⇒ weakPartialRefinement(pl, pl2, m))
      WHERE pl2 = (#F := F(pl), A := am2, K := ck2#)

removeAssetsSameProducts: THEOREM
  FORALL (pl, am2, ck2, s, its, pairs):
    ((s = ◇ (F(pl), K(pl), domain(pairs)) ∧
          syntaxAddAssets(am2, A(pl), ck2, K(pl), pairs, its) ∧
           conditionsAddAssets(pairs, its) ∧
            (FORALL c:
                 ¬ s(c) ⇒
                  SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
       ⇒
       (FORALL c:
            s(c) ⇒
             ((semantics(K(pl))(A(pl))(c)) =
                 (semantics(K(pl2))(A(pl2))(c)))))
      WHERE pl2 = (#F := F(pl), A := am2, K := ck2#)

removeAssetsPartRefStrong: THEOREM
  FORALL (pl, am2, ck2, s, its, pairs):
    ((s = ◇ (F(pl), K(pl), domain(pairs)) ∧
          syntaxAddAssets(am2, A(pl), ck2, K(pl), pairs, its) ∧
           conditionsAddAssets(pairs, its) ∧
            (FORALL c:
                    ¬ s(c) ⇒
                     SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
          ⇒ strongPartialRefinement(pl, pl2, s))
        WHERE pl2 = (#F := F(pl), A := am2, K := ck2#)

  removeAssetsPartRefWeak: THEOREM
    FORALL (pl, am2, ck2, m, its, pairs):
      ((domain(m) = ◇ (F(pl), K(pl), domain(pairs)) ∧
            syntaxAddAssets(am2, A(pl), ck2, K(pl), pairs, its) ∧
             conditionsAddAssets(pairs, its) ∧
              identity?(m) ∧
               (FORALL c:
                    ¬ domain(m)(c) ⇒
                     SPLrefinement.wfProduct(semantics(K(pl2))(A(pl2))(c))))
          ⇒ weakPartialRefinement(pl, pl2, m))
        WHERE pl2 = (#F := F(pl), A := am2, K := ck2#)

END SPLPartialRefTemplates
```