Appendix A: Further optimised AddAtom algorithm

This appendix documents the variation of the AddAtom algorithm with the best time performance. Other variations of the algorithm can be found in sections 4.3 and 4.6.

Note that for the sake of simplicity of the pseudo code, it is assumed that the context has no comparable objects and attributes and the objects and attributes of the context form the atoms and coatoms of the lattice. This is the case when FCA-lattices are equal to EA-lattices. Slight modifications and additional tests are necessary depending on which type of lattice is to be built.
Input context (0, A, I)
1_L = NewConcept(L)
0_L = NewConcept(L)
0_L.Intent = A
For ∀ a ∈ A
  aAttributeConcept = NewConcept(L)
  aAttributeConcept.Intent = {a}
  CreateArc(L, 0_L, aAttributeConcept)
  CreateArc(L, aAttributeConcept, 1_L)
Rof
For ∀ o ∈ O
  FastAddAtom (L, Intent(o), o, 0_L)
Rof

//===================================================================
Function GetMeet(anIntent, aConcept, attrCount)
Return aConcept
//===================================================================
// Determine the meet of anIntent by starting at Concept
parentIsMeet = True
Do While parentIsMeet
  parentIsMeet = False
  Parents = ConceptParents(L, aConcept)
  For ∀ Parent ∈ Parents
    If attrCount[Parent] = || anIntent || then
      aConcept = Parent
      parentIsMeet = True
      Exit For
    Fi
  Rof
Od
Return aConcept
End GetMeet

//===================================================================
Function AddAtomRecurse(L, anIntent, GeneratorConcept, attrCount,
ExactConcepts, DirtyConcepts, IgnoreConcepts)
Return aConcept
//===================================================================
CandidateParents = ConceptParents(L, GeneratorConcept)
ConceptParents = Ø
UCConceptParents = Ø
DCConceptParents = Ø
Exit = False
j = 0
// Concepts in CandidateParents that have the highest number of
// attributes of anIntent in their intents should be considered first
// and therefore sorted in decending order of the number of attrCount
For ∀ Candidate ∈ CandidateParents
  SortArray [j] = Candidate
  j = j + 1
Rof
Sort SortArray in decending order of attrCount[SortArray[j]]
For ∀ k = 0 to j - 1
  // Get candidate with next highest number of markers
  Candidate = SortArray[k]
  If (Candidate ∈ IgnoreConcepts) and (Candidate ∈ UCConceptParents)
    and (Candidate ∈ DCConceptParents) and Not Exit
    //Only Candidates with at least one attribute of anIntent should
    // be considered
newIntent = Candidate.Intent ∩ anIntent
Generator = GetMeet(L, newIntent, Candidate, attrCount, ExactConcepts, DirtyConcepts, IgnoreConcepts)

If Generator ∉ ExactConcepts then
    Generator = AddAtomRecurse(L, newIntent, Generator, attrCount, ExactConcepts, DirtyConcepts, IgnoreConcepts)
Fi
// At this point Generator is now an exact meet of anIntent
If newIntent = anIntent then
    Exit = True
Fi
If Generator ∉ UCConceptParents and not Exit then
    ConceptParents = ConceptParents ∪ {Generator}
    If attrCount[Generator] > 1 then
        // If the Generator is not an attribute we can remove the
        // concepts below and above it from consideration - this
        // is possible because all concepts that will be considered
        // hereafter will have a smaller attrCount
        UCCGenerator = UpwardClosure(L, Generator)
        ConceptParents = ConceptParents - UCCGenerator
        // Do not consider ConceptParents that are spanned by
        // Generator
        UCConceptParents = UCConceptParents ∪ UCCGenerator
        // Concepts above need not be considered
        DCGenerator = DownwardClosure(L, Generator)
        DCConceptParents = DCConceptParents ∪ DCGenerator
        // Concepts below it will not be considered
    Fi
Fi
Fi
Rof
NewConcept = CreateNewConcept(L)
NewConcept.Extent = GeneratorConcept.Extent
NewConcept.Intent = anIntent
attrCount [NewConcept] = ||anIntent||
ExactConcepts = ExactConcepts U {NewConcept}
For ∀ ConceptMeet ∈ ConceptParents
    If ConceptMeet in CandidateParents then
        DeleteArc(GeneratorConcept, ConceptMeet)
    Fi
CreateArc(NewConcept, ConceptMeet)
Rof
DeleteArc(GeneratorConcept, NewConcept)
Return NewConcept
End AddAtomRecurse

Function FastAddAtom(L, anIntent, o, GeneratorConcept)
//=============================================================================
DirtyAttrs = GetAttributes(L) - anIntent
DirtyConcepts = Ø
// DirtyConcepts: contains intents with attributes other than Intent
// and therefore all the approximate meets of anIntent
For ∀ attr ∈ DirtyAttrs
    DirtyConcepts = DirtyConcepts U DownwardClosure(L, attr)
// It is also possible to calculate DirtyConcepts using attrCount
Rof
CandidateConcepts = Ø
For ∀ attr ∈ anIntent
  CandidateConcepts = CandidateConcepts ∪ DownwardClosure(L, attr)

Rof
ExactConcepts = CandidateConcepts - DirtyConcepts
// ExactConcepts have only attributes of anIntent in their intents
// and form exact meets of subsets of anIntent
IgnoreConcepts = DirtyConcepts - CandidateConcepts
// IgnoreConcepts have no attributes of anIntent in their intents
// and can be ignored when searching for GeneratorConcepts

// Calculate markers: attrCount[Concept] is the number
// of attributes of anIntent for that concept (i.e. markers
// accumulated)
Let attrCount [x] = 0 for all x ∈ L
For ∀ Concept ∈ CandidateConcepts
  attrCount [Concept] = ||Concept.intent ∩ anIntent ||

Rof
NewConcept = AddAtom2(L, anIntent, EmptyConcept(L), GeneratorConcept, attrCount, ExactConcepts, DirtyConcepts, IgnoreConcepts)

For ∀ Concept ∈ UpwardClosure(L, NewConcept)
  Concept.Extent = Concept.Extent ∪ {g}

Rof
End FastAddAtom

//================================================================================
Appendix B: AddAtom algorithmic complexity bounds

The algorithm outline below show complexity bounds of the steps or group of steps for various parts of the AddAtom algorithm (documented in section 4.6). The complexity column indicates the complexity bound or alternatively the maximum number of iterations in the case of loops.
Function OptimisedAddAtom
  .
  For a
  .
  Rof
  For o
  .
  For x
  .
  Rof
  AddAtom()
  For x
  .
  Rof
  Rof
End OptimisedAddAtom

Function GetMeet()
  Do While ParentIsMeet
  For Parent ...
    If ...
      Fi
    Rof
  Od
End GetMeet

Function AddAtom()
  .
  For Candidate ...
  .
  If ...
    GetMeet()
    If
      AddAtom()
    Fi
  Else
    .
    Fi
    For g ...
      .
      Rof
  .
  For g ...
    .
    Rof
End AddAtom

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(\max(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(\max(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(1)$</td>
</tr>
<tr>
<td></td>
<td>$O(\max(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(\max(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(1)$</td>
</tr>
<tr>
<td></td>
<td>$O(\max(</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>