Xevolverによる変換サンプル

2014年11月5日

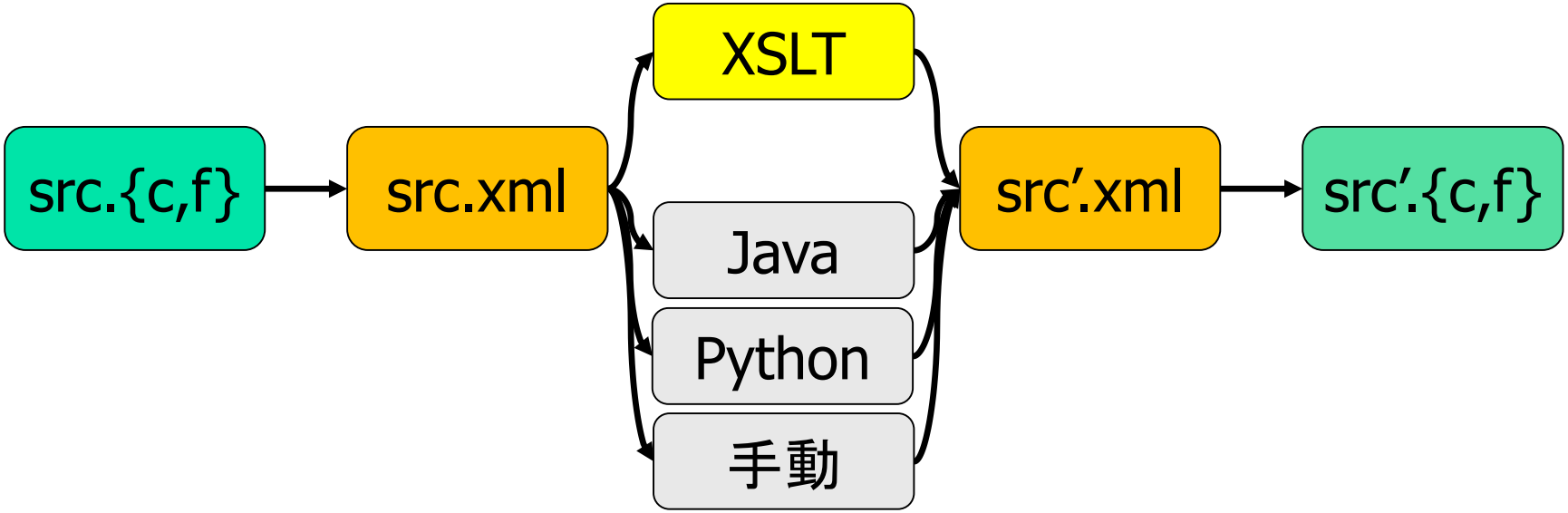
CRESTチームミーティング

東大理学部7号館511号室

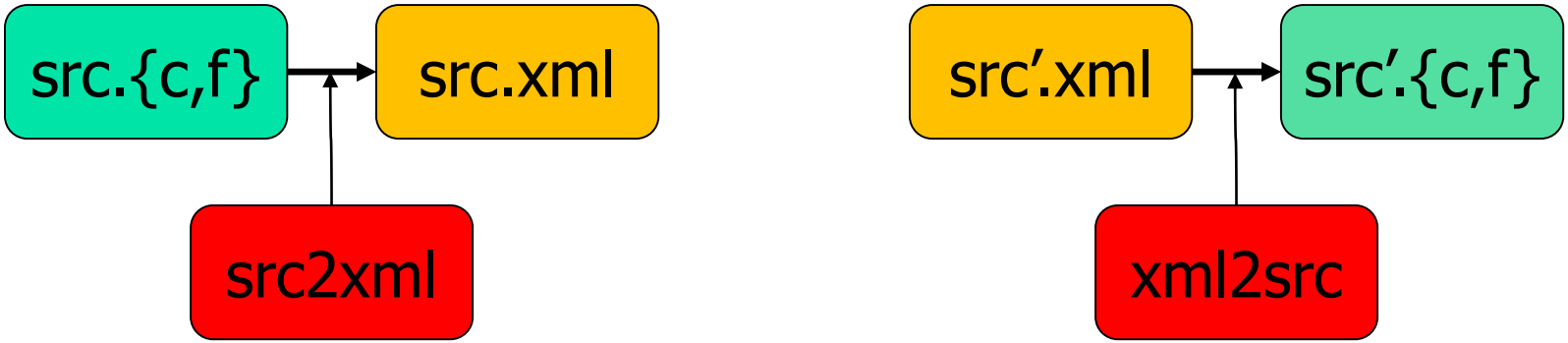
Xevolver概要

- プログラムソースを変換するためのフレームワーク
  - ≠プログラムソースを変換する

- プログラムソース(C,Fortran)をXMLに変換
  - (現状)ROSEのASTと一対一対応したXMLフォーマット
  - XMLを任意に操作することでプログラムソース変換を実現



src2xml, xml2src



ソースファイル

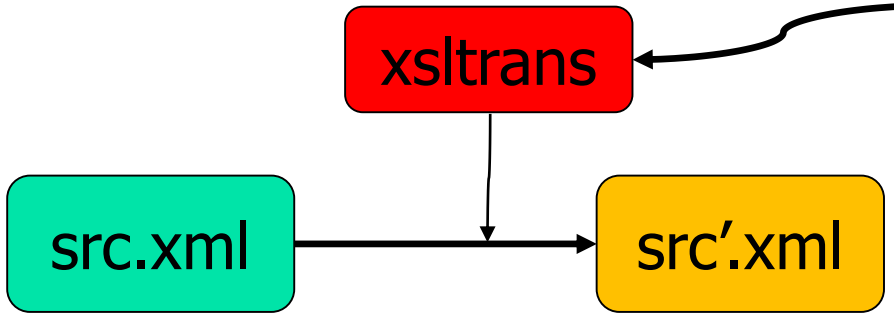
```
program test

  write(6,*) 'hello!'

!$xev dummy
end program test
```

XMLファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<SgSourceFile filename="hello.f90" language="4" format="2">
  <SgGlobal>
    <SgProgramHeader
      <SgTypeVoid/>
    <SgFunctionParar
    <SgFunctionDefinition>
      <SgBasicBlock>
        <SgWriteStatement fmt="true">
          <SgExprListExp>
            <SgStringVal value="hello!" SingleQuote="1" />
          </SgExprListExp>
          <SgIntVal value="6"  string="6" />
          <SgAsteriskShapeExp/>
        </SgWriteStatement>
<PreprocessingInfo pos="4"  type="3" >
!$xev dummy
</PreprocessingInfo>
<SgPragmaDeclaration >
  <SgPragma pragma="xev dummy" />
</SgPragmaDeclaration >
      </SgBasicBlock>
    </SgFunctionDefinition>
  </SgProgramHeaderStatement>
  </SgGlobal>
</SgSourceFile>
```

xsltrans(xslproc)

- 規格通りのXSLTプロセッサ



XSLファイル

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet
      xmlns:xs                                    ansform">
      <xsl:outp                                   8" />

      <xsl:template match="/">
            <xsl:apply-templates />
      </xsl:template>

      <xsl:template match="*">
            <xsl:copy>
                  <xsl:copy-of select="@*" />
                  <xsl:apply-templates />
            </xsl:copy>
      </xsl:template>

      <!-- remove PreprocessingInfo -->
      <xsl:template match="PreprocessingInfo">
            <xsl:comment>
                  PreprocessingInfo
            </xsl:comment>
            <!-- <xsl:apply-templates /> -->
      </xsl:template>

      <!-- remove SgPragmaDeclaration -->
      <xsl:template match="SgPragmaDeclaration">
            <xsl:comment>
                  SgPragmaDeclaration
            </xsl:comment>
      </xsl:template>

</xsl:stylesheet>
```

恒等変換を行うXSL

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output method="xml" encoding="UTF-8" />

      <xsl:template match="/">
            <xsl:apply-templates />
      </xsl:template>

      <xsl:template match="*">
            <xsl:copy>
                  <xsl:copy-of select="@*" />
                  <xsl:apply-templates />
            </xsl:copy>
      </xsl:template>

</xsl:stylesheet>
```

全マッチ

要素コピー

属性コピー

- xmlである宣言
- xslである宣言
- マッチ開始点
- 全コピー

最も単純な XSL Sample: 変数名変換

```
 <xsl:template match="SgVarRefExp">
   <xsl:choose>
     <xsl:when
test="ancestor::SgExprStatement/preceding::SgPragma/DIRECTIVE[@name='var']/CLAU
SE/@name='replace'">
       <xsl:copy> <!-- SgVarRefExp -->
         <xsl:attribute name="name">
         <xsl:value-of
select="ancestor::SgExprStatement/preceding::SgPragma/DIRECTIVE[@name='var']/CLA
USE[@name='replace']/ARG[2]/@value" />
         </xsl:attribute>
         <xsl:apply-templates></xsl:apply-templates>
       </xsl:copy>
     </xsl:when>
     <xsl:otherwise>
       <xsl:copy>
         <xsl:copy-of select="@*"></xsl:copy-of>
         <xsl:apply-templates></xsl:apply-templates>
       </xsl:copy>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:template>
```

SgVarRefExpにマッチ

replaceだったら

ディレクティブで指示された名前に変更する

XSL Sample: ループインターチェンジ

```
<xsl:template match="SgFortranDo" mode="loop_interchange">
  <xsl:choose>
   <xsl:when
    test="preceding-
sibling::*[1]/SgPragma/DIRECTIVE[@name='loop']/CLAUSE[@name='interchange']/ARG/
@value='1'">
    <xsl:element name="SgFortranDo">
     <xsl:copy-of select="SgBasicBlock/SgFortranDo/@*" />
     <xsl:copy-of select="SgBasicBlock/SgFortranDo/SgAssignOp" />
     <xsl:copy-of select="SgBasicBlock/SgFortranDo/SgIntVal" />
     <xsl:copy-of select="SgBasicBlock/SgFortranDo/SgNullExpression" />
     <xsl:element name="SgBasicBlock">
      <xsl:copy-of select="SgBasicBlock/@*" />
      <xsl:copy>
       <xsl:copy-of select="@*" />
       <xsl:copy-of select="./SgAssignOp" />
       <xsl:copy-of select="./SgIntVal" />
       <xsl:copy-of select="./SgNullExpression" />
       <xsl:copy-of select="SgBasicBlock/SgFortranDo/SgBasicBlock" />
      </xsl:copy>
     </xsl:element>
    </xsl:element>
   </xsl:when>
...
```

インターチェンジ用ルール

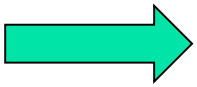
内側ループをコピー

その後、外側ループをコピー

変換Sample: NT-Opt (旧flatten)

- ベクトル向けコード → GPU向けコード

```
DO  M=1,MF
  DO  K=1,KF
    DO  J=1,JF
      !$xev nt_opt param(I, 1, inum)
      DO  L=Istart,Iend
        II1 = IS(L)
        II2 = II1+1
        II3 = II2+1
        IIF = IT(L)
        IIE = IIF-1
        IID = IIE-1
        DO 200 I=II2,IIF
          IF (I.LE.II3.OR.I.GE.IIE)THEN
            STBC=0.0D0
          ELSE
            STBC=1.0D0
          END IF
```

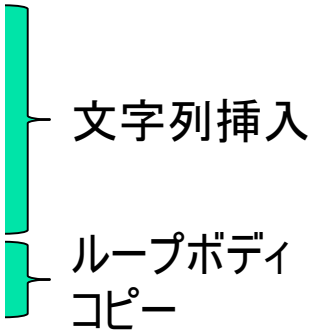


```
DO  M=1,MF
  DO  K=1,KF
    DO  J=1,JF
      DO I=1,inum
        DO L = Istart, Iend
          IF (I .GE. IS(L) .AND. I .LE. IT(L)) THEN
            EXIT
          END IF
        END DO
        IF (I.LE.II3.OR.I.GE.IIE)THEN
          STBC=0.0D0
        ELSE
          STBC=1.0D0
        END IF
```

XSL Sample: NT-Opt (旧flatten) その1

- 文字列挿入を用いた最短実装(一般性はない)
  - Lループの1レベル内側がIループであるという前提

```
<xsl:template match="SgFortranDo" mode="nt_opt">
 DO I=1,inum
 !$acc loop seq
 DO L=lstart,lend
 IF (I.ge.IS(L) .and. I.le.IT(L)) EXIT
 END DO
 <xsl:apply-templates select="SgBasicBlock/SgFortranDo/SgBasicBlock" />
 END DO
</xsl:template>
```



XSL Sample: NT-Opt (旧flatten) その2

- 再帰を用いた場合
  - LループとIループが離れても動作する一般性を持つ

```
<xsl:template match="*" mode="nt_opt">
 <xsl:choose>
  <xsl:when test="self::SgFortranDo/SgAssignOp/SgVarRefExp[@name='I']">
   DO <xsl:value-of select="self::SgFortranDo/@nlabel" /> I=1,inum
    !$acc loop seq
    DO L=lstart,lend
     IF (I.ge.IS(L) .and. I.le.IT(L)) EXIT
    END DO
    <xsl:apply-templates select="SgBasicBlock" />
  </xsl:when>
  <xsl:otherwise>
   <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates mode="nt_opt" />
   </xsl:copy>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Iループを見つけた

文字列挿入

ループボディコピー

間にあるループはコピー

参考: NT-Opt (旧flatten) その3

- 文字列挿入を使わず、一般性を持たせた場合、、、

```
<xsl:template match="SgFortranDo" mode="nt_opt">
 <xsl:param name="start" />
 <xsl:param name="end" />
 <xsl:copy>
  <xsl:copy-of select="SgBasicBlock/SgFortranDo/@*" /> <!--
SgFortranDo -->
  <xsl:element name="SgAssignOp">
   <xsl:copy-of
select="SgBasicBlock/SgFortranDo/SgAssignOp/SgVarRefExp[1]"
/>
   <xsl:element name="SgIntVal">
    <xsl:attribute name="value">
     <xsl:value-of select="$start" />
    </xsl:attribute>
   </xsl:element>
  </xsl:element>
  <xsl:element name="SgVarRefExp">
   <xsl:attribute name="name">
    <xsl:value-of select="$end" />
   </xsl:attribute>
  </xsl:element>
  <xsl:copy-of select="SgNullExpression" />
  <xsl:element name="SgBasicBlock">
   <xsl:element name="SgFortranDo">
    <xsl:copy-of select="@*" /> <!-- SgFortranDo →
.....
```

これ以外にルールif-filterが必要

```
....
    <xsl:element name="SgAssignOp">
     <xsl:element name="SgVarRefExp">
      <xsl:attribute name="name">
       <xsl:value-of select="SgAssignOp/SgVarRefExp[1]/@name" />
      </xsl:attribute>
     </xsl:element>
     <xsl:element name="SgVarRefExp">
      <xsl:attribute name="name">
       <xsl:value-of select="SgAssignOp/SgVarRefExp[2]/@name" />
      </xsl:attribute>
     </xsl:element>
    </xsl:element>
    <xsl:element name="SgVarRefExp">
     <xsl:attribute name="name">
      <xsl:value-of select="SgVarRefExp/@name" />
     </xsl:attribute>
    </xsl:element>
    <xsl:element name="SgNullExpression"></xsl:element>
    <xsl:element name="SgBasicBlock">
     <xsl:call-template name="if-filter">
      <xsl:with-param name="checkIndex" select="path to the index to filter" />
      <xsl:with-param name="arrayStart" select="path to the array of start indices" />
      <xsl:with-param name="arrayEnd" select="path to the array of end indices" />
      <xsl:with-param name="arrayIndex" select="path to the index name of the array
of start and end" />
     </xsl:call-template>
     <xsl:apply-templates
select="SgBasicBlock/SgFortranDo/SgBasicBlock/SgExprStatement" />
    </xsl:element>
   </xsl:element>
  </xsl:element>
 </xsl:copy>
</xsl:template>
```

複合変換: NT-Opt + OpenACC挿入

- 範囲を指定した複合変換を開始する

```
 <xsl:template match="SgFortranDo">
  <xsl:choose>
   <xsl:when
    test="preceding-sibling::SgPragmaDeclaration/SgPragma/DIRECTIVE/@name  = 'nt_opt'">
   <xsl:apply-templates select="self::SgFortranDo"
     mode="nt_opt">
     <xsl:with-param name="label"
      select='preceding-
sibling::SgPragmaDeclaration/SgPragma/DIRECTIVE/CLAUSE/ARG/@value' />
     <xsl:with-param name="depth" select='1' />
    </xsl:apply-templates>
   </xsl:when>
…
```

つづく

NT-Opt with OpenACC

```
<xsl:template match="SgFortranDo" mode="nt_opt">
 <xsl:param name="label" />
 <xsl:param name="depth" />
 <xsl:call-template name="add-openacc">
  <xsl:with-param name="label" select="$label" />
  <xsl:with-param name="depth" select="$depth" />
 </xsl:call-template>
 <xsl:choose>
  <xsl:when test="SgAssignOp/SgVarRefExp/@name='L'">
   <xsl:apply-templates select="SgBasicBlock/SgFortranDo"
    mode="nt_opt">
    <xsl:with-param name="label" select="$label" />
    <xsl:with-param name="depth" select="$depth+1" />
   </xsl:apply-templates>
  </xsl:when>
  <xsl:when test="SgAssignOp/SgVarRefExp/@name='I'">
   DO <xsl:value-of select="$label" /> I=1,inum
   !$acc loop seq
   DO L=lstart,lend
   IF (I.ge.IS(L) .and.
   I.le.IT(L)) EXIT
   END DO
   END DO
   <xsl:apply-templates select="SgBasicBlock" />
  </xsl:when>
  <xsl:otherwise>
   <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates mode="nt_opt">
     <xsl:with-param name="label" select="$label" />
     <xsl:with-param name="depth" select="$depth+1" />
    </xsl:apply-templates>
   </xsl:copy>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

OpenACC
ディレクティブを追加するルール
(次スライド)

ユーザ定義ルール: OpenACCディレクティブ挿入

```
<xsl:template name="add-openacc">
 <xsl:param name="label" />
 <xsl:param name="depth" />
 <xsl:choose>
  <xsl:when test="$label = 200">
   <xsl:choose>
    <xsl:when test="$depth = 1">
     <xsl:text>!$acc loop private(L)</xsl:text>
    </xsl:when>
    <xsl:when test="$depth = 2">
     <xsl:text>!$acc loop gang</xsl:text>
    </xsl:when>
    <xsl:when test="$depth = 3">
     <xsl:text>!$acc loop gang,vector</xsl:text>
    </xsl:when>
    <xsl:when test="$depth = 4">
     <xsl:text>!$acc loop vector</xsl:text>
    </xsl:when>
   </xsl:choose>
  </xsl:when>
  <xsl:when test="$label = 300">
  </xsl:when>
 </xsl:choose>
</xsl:template>
```

ディレクティブのパターンごとに用意する

XSL Sample: ライブラリ呼び出しによる変換

```
!$xev loop_tag
  do k=1,n-1
    do j=1,n-1
      do i=1,n-1
        B(i,j,k) = A(i,j,k)
      end do
    end do
  end do
```

1. Unroll_Jam
2. Unroll
3. Unroll_Jam + Unroll

Unroll_Jam

```
<xsl:template match="SgFortranDo">
<xsl:choose>
<xsl:when test="preceding-sibling::*[1]/SgPragma/@pragma = 'xev loop_tag'">
<xsl:apply-templates select="." mode="chill_unroll_jam">
<xsl:with-param name="max" select="4" />
<xsl:with-param name="var" select="'k'" />
</xsl:apply-templates>
</xsl:when>

<xsl:otherwise>
<xsl:copy>
<xsl:copy-of select="@*" />
<xsl:apply-templates />
</xsl:copy>
</xsl:otherwise>

</xsl:choose>
</xsl:template>
```

```
PROGRAM triple_loop_1
  INTEGER, PARAMETER :: n = 139
  REAL :: A(n,n,n), B(n,n,n)
  DO k = 1, n - 1, 4
    DO j = 1, n - 1
      DO i = 1, n - 1
        B(i,j,k) = A(i,j,k)
        B(i,j,k + 1) = A(i,j,k + 1)
        B(i,j,k + 2) = A(i,j,k + 2)
        B(i,j,k + 3) = A(i,j,k + 3)
      END DO
    END DO
  END DO
END PROGRAM
```

- 具体的な変換はライブラリにおまかせ
- パラメータ(段数とループ名)のみ指定

Unroll

```
<xsl:template match="SgFortranDo">
<xsl:choose>
<xsl:when test="preceding-sibling::*[1]/SgPragma/@pragma = 'xev loop_tag'">
<xsl:comment>
test-2.xsl xev loop_tag
</xsl:comment>
<xsl:apply-templates select="." mode="find_loop" />
</xsl:when>

<xsl:otherwise>
<xsl:copy>
<xsl:copy-of select="@*" />
<xsl:apply-templates />
</xsl:copy>
</xsl:otherwise>

</xsl:choose>
</xsl:template>

<xsl:template match="*" mode="find_loop">
<xsl:choose>
<xsl:when test="self::SgFortranDo/SgAssignOp/SgVarRefExp/@name = 'i'">
<xsl:apply-templates select="." mode="chill_unroll">
<xsl:with-param name="max" select="4" />
<xsl:with-param name="var" select="'i'" />
</xsl:apply-templates>
</xsl:when>
<xsl:otherwise>
<xsl:copy>
<xsl:copy-of select="@*" />
<xsl:apply-templates mode="find_loop" />
</xsl:copy>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

ディレクティブで指定された範囲を検出する

```
PROGRAM triple_loop_1
  INTEGER, PARAMETER :: n = 139
  REAL :: A(n,n,n), B(n,n,n)
  DO k = 1, n - 1
    DO j = 1, n - 1
      DO i = 1, n - 1, 4
        B(i,j,k) = A(i,j,k)
        B(i + 1,j,k) = A(i + 1,j,k)
        B(i + 2,j,k) = A(i + 2,j,k)
        B(i + 3,j,k) = A(i + 3,j,k)
      END DO
    END DO
  END DO
END PROGRAM
```

変換したいループまで移動して、ライブラリ呼び出し

Unroll_Jam + Unroll

```
<xsl:template match="SgFortranDo">
<xsl:choose>
<xsl:when test="preceding-sibling::*[1]/SgPragma/@pragma = 'xev loop_tag'">
<xsl:comment>
test-3.xsl xev loop_tag
</xsl:comment>

<xsl:variable name="step1">
<xsl:apply-templates select="." mode="chill_unroll_jam">
<xsl:with-param name="max" select="4" />
<xsl:with-param name="var" select="'k'" />
</xsl:apply-templates>
</xsl:variable>

<xsl:apply-templates select="exslt:node-set($step1)"
mode="find_loop_and_unroll" />
</xsl:when>

<xsl:otherwise>
<xsl:copy>
<xsl:copy-of select="@*" />
<xsl:apply-templates />
</xsl:copy>
</xsl:otherwise>

</xsl:choose>
</xsl:template>
```

Unroll_Jam

Unroll

まとめ

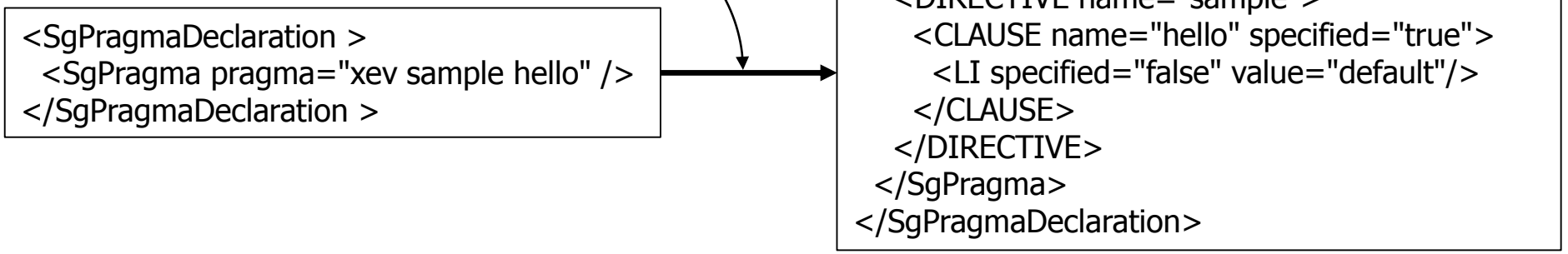
- Xevolverの概要
- 各コマンドの動作
- XSLTによる変換ルールのサンプル
  - 恒等変換
  - 単一ルールによる単純な変換
  - 文字列挿入による変換
    - ↔ 一般性を確保した変換
  - 複合ルールを用いた変換
  - ライブラリを用いた変換

Appendix: dir2xml

- ディレクティブ文字列をXMLにパースする

```
<DIRECTIVE name="sample">
  <CLAUSE name="hello">
    <LI value="default" />
  </CLAUSE>
</DIRECTIVE>
```

```
<SgPragmaDeclaration >
  <SgPragma pragma="xev sample hello" />
</SgPragmaDeclaration >
```



```
<SgPragmaDeclaration>
  <SgPragma pragma="xev sample hello">
    <DIRECTIVE name="sample">
      <CLAUSE name="hello" specified="true">
        <LI specified="false" value="default"/>
      </CLAUSE>
    </DIRECTIVE>
  </SgPragma>
</SgPragmaDeclaration>
```