

Allomorphie in JSLIM

Johannes Handl

Department Germanistik und Komparatistik
Professur für Computerlinguistik
Friedrich-Alexander Universität Erlangen

07.05.08

Teil I

Status quo ante

Lexikondatei

```
# starke Deklination 1u (Bach)
!template[ allo: a_bach
           flx:  C_bach
           cat:  ()
           sem:  ()
]

!+sur core: Bach
           Baum
           Fuchs
           Sohn
           ...
```

Regeldatei

```
# starke Deklination 1u (Bach)
RULE_DEKLINATION_STARK_BACH

[sur: /(.*)(a|o|u)(.*)/, allo: a_bach]

pcopy(P)
regset($1 $2 $3 P.sur)
ecopy("yes" P.nochange)
ecopy("m" P.cat)
ecopy("s3+a" P.sem)
result(P)
...
pcopy(P)
umlaut($2)
regset($1 $2 $3 P.sur)
ecopy("pl" P.cat)
result(P)
```

Weitere Dateien

Es werden die folgenden weiteren Dateien benötigt:

- **Attributdatei**
- Variablendatei
- Operationsdatei
- Projektdatei

Weitere Dateien

Es werden die folgenden weiteren Dateien benötigt:

- Attributdatei
- Variablendatei
- Operationsdatei
- Projektdatei

Weitere Dateien

Es werden die folgenden weiteren Dateien benötigt:

- Attributdatei
- Variablendatei
- Operationsdatei
- Projektdatei

Weitere Dateien

Es werden die folgenden weiteren Dateien benötigt:

- Attributdatei
- Variablendatei
- Operationsdatei
- Projektdatei

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Desiderata

- Kompaktere Regeln
- Deklarative Syntax
- Lemma als Dreh- und Angelpunkt der Regelausführung
- Explizite Kodierung von Gemeinsamkeiten (Vererbung)
- Gezielte Generierung von Allomorphen
- Gezielte Aktualisierung von Allomorphlexika
- Generierung von Templates
- Mehr Effizienz
- Möglichst wenig Java-Code

Teil II

Status quo

Lexikondatei

```
!template[ allo: a_bach  
           flx: C_bach  
           cat: ()  
           sem: ()  
]
```

```
!+sur: Bach  
      Baum  
      Fuchs  
      Sohn  
      ...
```



```
!template[ allo: A_bach  
           flx: C_bach  
]
```

```
!+sur: B|a|ch  
      B|a|um  
      F|u|chs  
      S|o|hn  
      ...
```

Lexikondatei

```
!template[ allo: a_bach  
           flx: C_bach  
           cat: ()  
           sem: ()  
]
```

```
!+sur: Bach  
      Baum  
      Fuchs  
      Sohn  
      ...
```



```
!template[ allo: A_bach  
           flx: C_bach  
]
```

```
!+sur: B|a|ch  
      B|a|um  
      F|u|chs  
      S|o|hn  
      ...
```

Regeldatei

```
# starke Deklination 1u (Bach)  
RULE_DEKLINATION_STARK_BACH
```

```
[sur: /(.*)(a|o|u)(.*)/, allo: a_bach]
```

```
pcopy(P)  
regset($1 $2 $3 P.sur)  
ecopy(m-g P.cat)  
ecopy(m sg P.sem)  
result(P)
```

```
pcopy(P)  
umlaut($2)  
regset($1 $2 $3 P.sur)  
ecopy(pstem P.cat)  
ecopy(pl P.cat)  
result(P)
```

→

```
table A_Bach: [sur] => [sur,cat,sem]  
/(.*)\|([AOUaou])\|(.*)/  
=> /$1$2$3/, (m-g), (m sg) ;  
=> /$1$2e$3/, (pstem), (m pl) .
```

Regeldatei

```
# starke Deklination 1u (Bach)  
RULE_DEKLINATION_STARK_BACH
```

```
[sur: /(.*)(a|o|u)(.*)/, allo: a_bach]
```

```
pcopy(P)  
regset($1 $2 $3 P.sur)  
ecopy(m-g P.cat)  
ecopy(m sg P.sem)  
result(P)
```

```
pcopy(P)  
umlaut($2)  
regset($1 $2 $3 P.sur)  
ecopy(pstem P.cat)  
ecopy(pl P.cat)  
result(P)
```

→

```
table A_Bach: [sur] => [sur,cat,sem]  
/(.*)\|([AOUaou])\|(.*)/  
=> /$1$2$3/, (m-g), (m sg) ;  
=> /$1$2e$3/, (pstem), (m pl) .
```

Weitere Dateien

Weiteren Dateien:

- Attributdatei
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

Weitere Dateien

Weiteren Dateien:

- **Attributdatei**
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

Weitere Dateien

Weiteren Dateien:

- Attributdatei
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

Weitere Dateien

Weiteren Dateien:

- Attributdatei
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

Weitere Dateien

Weiteren Dateien:

- Attributdatei
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

Weitere Dateien

Weiteren Dateien:

- Attributdatei
- Keine Variablendatei
- Keine Operationsdatei
- Bedingt Projektdatei

'The gory details'

```
table A_Bach: [sur] => [sur,cat,sem]
/(.*)\|([AOUaou])\|(.*)/
=> /$1$2$3/, (m-g), (m sg) ;
=> /$1$2e$3/, (pstem),(m pl) .
```

- Trie
- Vererbung
- Common Subtree Sharing
- Autocompletion

'The gory details'

```
table A_Bach: [sur] => [sur,cat,sem]
/(.*)\|([AOUaou])\|(.*)/
=> /$1$2$3/, (m-g), (m sg) ;
=> /$1$2e$3/, (pstem),(m pl) .
```

- Trie
- Vererbung
- Common Subtree Sharing
- Autocompletion

'The gory details'

```
table A_Bach: [sur] => [sur,cat,sem]
/(.*)\|([AOUaou])\|(.*)/
=> /$1$2$3/, (m-g), (m sg) ;
=> /$1$2e$3/, (pstem),(m pl) .
```

- Trie
- Vererbung
- Common Subtree Sharing
- Autocompletion

'The gory details'

```
table A_Bach: [sur] => [sur,cat,sem]  
/(.*)\|([AOUaou])\|(.*)/  
=> /$1$2$3/, (m-g), (m sg) ;  
=> /$1$2e$3/, (pstem),(m pl) .
```

- Trie
- Vererbung
- Common Subtree Sharing
- Autocompletion

ObjectTrieNode

```
public class ObjectTrieNode implements Cloneable{
    final public ObjectTrieEntry value;
    final protected Object[] labels;
    final protected ObjectTrieNode[] targets;

    public ObjectTrieNode(ObjectTrieEntry value, Object[] labels,
        ObjectTrieNode[] targets){
        this.value = value;
        this.labels = labels;
        this.targets = targets;
    }
    public ObjectTrieNode clone(){ ... }
    protected ObjectTrieNode(ObjectTrieNode node){...}
    public ObjectTrieNode extend(ObjectTrieNode node){ ...}
    protected int find(Object[] data, Object key){ ... }
}
```

ObjectTrie

```
public class ObjectTrie extends ObjectTrieNode {  
  
    public ObjectTrie(ObjectTrieEntry value, Object[] labels,  
                      ObjectTrieNode[] targets){  
        super(value,labels,targets);  
    }  
    public ObjectTrie clone(){ ... }  
    public ObjectTrie insert(Object[] keys, int keylen, ObjectTrieEntry entry){  
  
    public ObjectTrieEntry lookup(Object[][] keys, int count) { ... }  
    ...  
}
```

ITableVal

```
public interface ITableVal extends IGetName, IVal, IXmlProject {
    TableSignatur getSignatur();
    int key(IVal[] [] proplet, int count, Object[] [] key);
    int instanceKey(IVal[] template, IVal[] instance, Object[] [] key);
    int propletKey(IVal[] proplet, Object[] [] key);
    CategorialOperation[] lookup(Object[] [] key, int count);
    ObjectTrie getTrie();
    ObjectTrie getReverseTrie();
    ...
}
```

CategorialOperation

```
public class CategorialOperation {
    final public LAGProject lagProject;
    final private int[] proplets;
    final private int[] [] attributes;
    final private TableTransitionAction[] [] actions;
    final private IMulticat[] multicats;

    protected CategorialOperation(
        LAGProject lagProject, int[] proplets, int[] [] attributes,
        TableTransitionAction[] [] actions, IMulticat[] multicats){
        this.lagProject = lagProject;
        this.proplets = proplets;
        this.attributes = attributes;
        this.actions = actions;
        this.multicats = multicats;
    }
    ...
}
```

CategorialOperation

```
public class CategorialOperation {  
    ...  
    public void execute(final IVal[] [] vals, int num, final IVal multicatval){  
        // iterate over the indices of the proplets to modify  
        for (int i=0;i<proplets.length;++i){  
            // get array of attributes  
            IVal[] attrs = vals[proplets[i]];  
            // iterate over the attributes indices of the proplets to modify  
            for(int j=0;j<attributes[i].length;++j){  
                // perform action  
                actions[i][j].execute(attrs,j);  
            }  
        }  
        // add the multicats the allocation, so that they can be found  
        for(IMulticat mc:multicats){  
            multicatval.setChild(mc.getId(),mc);  
        }  
    }  
    ...  
}
```

CategorialOperation

```
public class CategorialOperation {  
    ...  
    public void execute(IVal[] attrs, final IVal multicatval){  
        // iterate over the attributes indices of the proplets to modify  
        for(int j=0;j<attributes[0].length;++j){  
            // perform action  
            actions[0][j].execute(attrs,attributes[0][j]);  
        }  
        // add the multocats the allocation, so that they can be found  
        for(IMulticat mc:multicats){  
            multicatval.setChild(mc.getId(),mc);  
        }  
    }  
    ...  
}
```

CategorialOperation

```
public class CategorialOperation {
    public void executeIfSet(IVal[] attrs, final IVal multicatval){
        // iterate over the attributes indices of the proplets to modify
        for(int j=0;j<attributes[0].length;++j){
            int index = attributes[0][j];
            if(attrs[index] != null){
                // perform action
                actions[0][j].execute(attrs,index);
            }
        }
    }
    public void executeIfPossible(IVal[] attrs, final IVal multicatval){
        // iterate over the attributes indices of the proplets to modify
        for(int j=0;j<attributes[0].length;++j){
            int index = attributes[0][j];
            if(attrs[index] != null || !actions[0][j].READ){
                // perform action
                actions[0][j].execute(attrs,index);
            }
        }
    }
}
```

TableTransitionAction

```
public abstract class TableTransitionAction{
    public final boolean READ;

    public abstract void execute(IVal[] attrs, int index);

    public TableTransitionAction(boolean read){
        this.READ = read;
    }
}
```

TableTransitionAction

```
class TableTransitionRegexpAction extends TableTransitionAction {
    final public String from,to;
    public TableTransitionRegexpAction(String from,String to){
        super(true);
        this.from = from;
        this.to = to;
    }
    @Override
    final public void execute(final IVal[] attrs, final int index) throws JSL
        // get string attribute vaue to modify
        IStringVal sval = attrs[index].flatCopy().getIStringVal();
        // get string value
        String str = sval.getString();
        // perform substitution
        str = str.replaceFirst(from,to);
        sval.setString(str);
        // store modified value
        attrs[index] = sval;
    }
    ...
}
```

TableTransitionListSetAction

```
public class TableTransitionListSetAction extends TableTransitionAction {
    final public IVal[] leftvals;
    final public IVal[] rightvals;
    final public int left;
    final public int right;

    public TableTransitionListSetAction(int left, int right,
                                       IVal[] leftvals, IVal[] rightvals){
        super(true);
        this.leftvals = leftvals;
        this.rightvals = rightvals;
        this.left = left;
        this.right = right;
    }
    ...
}
```

TableTransitionListSetAction

```
public class TableTransitionListSetAction extends TableTransitionAction {
    ...
    @Override
    public void execute(IVal[] attrs, int index) {
        // get list
        IListVal listVal = attrs[index].getIListVal();
        // prepare list value for modification
        listVal.prepare();
        // remove on the left side
        for(int k=0;k<left;++k)listVal.remove(0);
        // remove on the right side
        for(int k=0, len = listVal.size()-1;k<right;++k)listVal.remove(len--);
        // add on the left side of a list (reverse)
        for(int k=leftvals.length-1;k>=0;--k)listVal.add(0,leftvals[k]);
        // add on the right side
        for(IVal v:rightvals)listVal.add(v);
        // commit changes
        attrs[index] = listVal.commit();
    }
    ...
}
```

Teil III

Ausblick

Linguistische Fragestellungen

- Strukturierung des Lexikons
 - nach Allo-Klassen
 - nach Kombi-Klassen
 - Karthesisches Produkt aus Alloklassen und Kombiklassen
- Information in Basislexikon
 - Grammatikalische Kategorien bereits im Basislexikon oder Generierung durch Alloregeln?
 - Zusätzliches Lexikon mit wortspezifischer Information?
 - Alloregel für alle Einträge?
- Umlaute
 - Kodierung im Lexikon?
 - Mittels gewöhnlicher regulärer Ausdrücke?
 - Mittels linguistisch erweiterter regulärer Ausdrücke?

Linguistische Fragestellungen

- Strukturierung des Lexikons
 - nach Allo-Klassen
 - nach Kombi-Klassen
 - Karthesisches Produkt aus Alloklassen und Kombiklassen
- Information in Basislexikon
 - Grammatikalische Kategorien bereits im Basislexikon oder Generierung durch Alloregeln?
 - Zusätzliches Lexikon mit wortspezifischer Information?
 - Alloregel für alle Einträge?
- Umlaute
 - Kodierung im Lexikon?
 - Mittels gewöhnlicher regulärer Ausdrücke?
 - Mittels linguistisch erweiterter regulärer Ausdrücke?

Linguistische Fragestellungen

- Strukturierung des Lexikons
 - nach Allo-Klassen
 - nach Kombi-Klassen
 - Karthesisches Produkt aus Alloklassen und Kombiklassen
- Information in Basislexikon
 - Grammatikalische Kategorien bereits im Basislexikon oder Generierung durch Alloregele?
 - Zusätzliches Lexikon mit wortspezifischer Information?
 - Alloregele für alle Einträge?
- Umlaute
 - Kodierung im Lexikon?
 - Mittels gewöhnlicher regulärer Ausdrücke?
 - Mittels linguistisch erweiterter regulärer Ausdrücke?

Linguistische Fragestellungen

- Diminutiv
 - Berücksichtigung bei Einteilung in Alloklassen
 - Berücksichtigung durch eigenes Attribut

Programmiertechnische Fragestellungen

- Erweiterung der Tabellen-Syntax um Platzhalter für das Verschieben von Attributen?
- Verwendung eines ausgezeichneten Nullwerts für das Löschen eines Attributs?
- Art der Integration der Allomorphie, beziehungsweise bestehender Tools in die Projektdatei.
- Syntax für die gezielte Generierung von Allomorphen und Wortformen.
- Deklarative Syntax auch auf Regeln von LA-Hear anwendbar?

Programmiertechnische Fragestellungen

- Erweiterung der Tabellen-Syntax um Platzhalter für das Verschieben von Attributen?
- Verwendung eines ausgezeichneten Nullwerts für das Löschen eines Attributs?
- Art der Integration der Allomorphie, beziehungsweise bestehender Tools in die Projektdatei.
- Syntax für die gezielte Generierung von Allomorphen und Wortformen.
- Deklarative Syntax auch auf Regeln von LA-Hear anwendbar?

Programmiertechnische Fragestellungen

- Erweiterung der Tabellen-Syntax um Platzhalter für das Verschieben von Attributen?
- Verwendung eines ausgezeichneten Nullwerts für das Löschen eines Attributs?
- Art der Integration der Allomorphie, beziehungsweise bestehender Tools in die Projektdatei.
- Syntax für die gezielte Generierung von Allomorphen und Wortformen.
- Deklarative Syntax auch auf Regeln von LA-Hear anwendbar?

Programmiertechnische Fragestellungen

- Erweiterung der Tabellen-Syntax um Platzhalter für das Verschieben von Attributen?
- Verwendung eines ausgezeichneten Nullwerts für das Löschen eines Attributs?
- Art der Integration der Allomorphie, beziehungsweise bestehender Tools in die Projektdatei.
- Syntax für die gezielte Generierung von Allomorphen und Wortformen.
- Deklarative Syntax auch auf Regeln von LA-Hear anwendbar?

Programmiertechnische Fragestellungen

- Erweiterung der Tabellen-Syntax um Platzhalter für das Verschieben von Attributen?
- Verwendung eines ausgezeichneten Nullwerts für das Löschen eines Attributs?
- Art der Integration der Allomorphie, beziehungsweise bestehender Tools in die Projektdatei.
- Syntax für die gezielte Generierung von Allomorphen und Wortformen.
- Deklarative Syntax auch auf Regeln von LA-Hear anwendbar?