ALE

Het programma ALE.PL is te vinden in directory `/users1/koeling/ale`. Hier staan ook een paar voorbeelden en de handleiding in 'tex'-formaat.

De nieuwste versie van ALE is per anonymous FTP op te halen op het volgende adres:

`ftp://j.gp.cs.cmu.edu/usr1/carp/ftp/`

De handleiding ligt ook ter inzage op kamer 4434.

ALE - grammatica's

Een grammatica in ALE kan de volgende onderdelen bevatten:

+ Type-definitie	
+ Type-definitie	
- Constraints op types	(→ opmerking)
+ Lexicon	
- Lexical Rules	
+ Grammar Rules	
- Definite Clauses	
- Templates (Macro's)	(→ voorbeeld)

Nu kijken we slechts naar die onderdelen die minimaal nodig zijn om een grammatica in ALE te kunnen schrijven. Deze onderdelen zijn hierboven gemarkeerd met een +.

Voordat we deze onderdelen kunne bekijken aan de hand van een voorbeeld, moeten we eerst kijken naar de eisen die ALE stelt aan de type-definitie. Zolang de type-definitie niet in orde is, kunnen we nog niets beginnen.

De Type-hierarchie:

ALE heet sterk getypeerd te zijn. Dit houdt in dat elke structuur voorzien moet zijn van een type. Types zijn georganiseerd in een type-hierarchy. Dit houdt in dat voor een bepaald type subtypes gedefinieerd kunnen zijn. Deze subtypes erven de beperkingen die opgelegd zijn aan de types die boven hen in de type-hierarchy staan (inheritance-based polymorphism). Tussen de types en hun subtypes bestaat een transitieve relatie (als b een subtype is van a en c van b, dan is c een subtype van a). Deze hoeft echter niet door de gebruiker expliciet gemaakt te worden, de compiler berekent de transitieve afsluiting.

Restricties op de type-definitie:

- De afgeleide transitieve relatie moet anti-symmetrisch zijn D.w.z.: het mag niet zo zijn dat twee verschillende types elkaar als subtype hebben. (partiele ordening)
- Het meest algemene type moet altijd het type *bot* zijn (elk type is een subtype van *bot*).
- De type-hierarchie moet *bounded complete* zijn. D.w.z.: Elk paar types dat een gemeenschappelijk subtype heeft, moet tevens een unieke meest algemene gemeenschappelijk subtype hebben. Bijvoorbeeld:

```
bot sub [a,b].
          a sub [c,d].
            c sub [].
            d sub [].
          b sub [c,d].
```

  voldoet niet aan deze eis, daar *c* en *d* allebei wel subtypes zijn van zowel *a* als *b*, maar er niet een meest algemene gemeenschappelijk subtype is. Wel goed is als de type-definitie als volgt wordt aangepast:

```
bot sub [a,b].
  a sub [e].
    e sub [c,d].
      c sub [].
      d sub [].
  b sub [e].
```

  Nu is *e* de meest algemene subtype van zowel *a* als *b*.

ALE - commando's

Opstarten (vanuit Sicstus):

```
?- compile(ale).
?- compile\_gram(GramFile).
```

Wanneer de grammatica groot begint te worden (wat bv. aan de hand is bij de HPSG-grammatica), kan het volgende erg nuttig zijn:

De eerste keer moet je `compile_gram` gebruiken, wanneer je je type-systeem niet meer veranderd, kan je ook alleen de grammatica (of onderdelen daarvan) opnieuw compileren m.b.v. `compile_grammar(`$GramFile$`)` (of `compile_lex/` `compile_rules(`$GramFile$`)`)

Debuggen :

Handige hulpmiddelen zijn:

(Merk op dat al deze functies (door gebruik van variabelen) ook gebruikt kunnen worden om mogelijke oplossingen te genereren)

Testen/debuggen van type-definitie:

`?- type(`$< Type >$`).`	$\Longrightarrow$ Test of $< Type >$ een type is.
`?- feature(`$< Feature >$`).`	$\Longrightarrow$ ....
`?- introduce(`$< F >$`,`$< T >$`).`	$\Longrightarrow$ Test of feature $< F >$ wordt geintroduceerd door type $< T >$
`?- approp(`$< F >$`,`$< T >$`,`$< Restr >$`).`	$\Longrightarrow$ Feature $< F >$ geintroduceerd bij type $< T >$ moet van type $< Restr >$ zijn.
`?- show_type` $< Type >$`.`	$\Longrightarrow$ Geeft plaats van type in type-hierarchy

Beperken van de output (de feature structuren kunnen erg groot worden):

`?- no_write_type(`$< T >$`).`	$\Longrightarrow$ Het schrijven van type $< T >$ wordt onderdrukt
`?- write_type(`$< T >$`).`	$\Longrightarrow$ ...
`?-(no_)write_feature(`$< F >$`).`	$\Longrightarrow$ idem

Grammatica bekijken:

`?- lex(`$< Woord >$`).`	$\Longrightarrow$ Laat FS van een 'lexical entry' zien.
`?- rule` $< Rule >$`.`	$\Longrightarrow$ ....
`?- macro` $< Macro(\_,\_,...) >$`.`	$\Longrightarrow$ ....

Werken met de grammatica:

`?- rec [mia,dances].`	$\Longrightarrow$ Parse de string 'mia dances'.

Als een string geaccepteerd is, is de analyse van een deelstring te bekijken door:

`?- edge(0,2).`	$\Longrightarrow$ Bekijk de deelstring bestaande uit de eerste twee woorden.

Het parseer-proces is op de voet te volgen door de mini-interpreter op te starten:

`?- interp.` (uit te schakelen door `?- nointerp.`)

De compiler stopt nu op het moment dat hij een 'edge' in de 'chart' toe kan voegen. Steeds geeft hij weer wat er toegevoegd gaat worden en ten gevolge van welke regel dat gebeurd. De gebruiker kan aangeven dat deze 'edge' toegevoegd kan worden ('add') en dat deze 'edge' niet toegevoegd moet worden ('noadd').