**Compiladors**

**Departament de Llenguatges i Sistemes Informàtics**

**Facultat d'Informàtica**

# Pràctica

# Analitzador d'*Erlang*

**Albert Bellonch**

Tardor 2010/2011

# Índex

# 1 Propòsit i visió general

L'objectiu d'aquest compilador és de fer d'analitzador lèxic, sintàctic i semàntic del llenguatge de programació Erlang, de propòsit general i que té els subconjunts sequencial (on és funcional, com Haskell o Miranda) i concurrent (on segueix el model *actor*, de forma que a partir d'un missatge rebut, un procés pren decisions, crea més processos, envia missatges i, en general, fa quelcom en resposta al propi missatge rebut).

L'objectiu final d'aquest analitzador és mostrar un gràfic, basat en el fitxer font escrit en Erlang, que reflecteixi la dependència entre processos i funcions d'aquest, així com altres elements a destacar-ne. Així doncs, tenim dues fases ben diferenciades:

| Part | Explicació | Tipus d'anàlisi |
|---|---|---|
| 1 | Anàlisi del fitxer font i conversió en un arbre sintàctic abstracte (**AST**, Abstract Syntax Tree). Aquest pas es fa per mitjà de l'ús del programa **PCCTS**, que se serveix del generador de parsers **ANTLR** i el generador d'analitzadors lèxics **DLG** entre altres. Aquí es detectarà si el codi font és vàlid lèxica i sintàcticament, i si és així es construirà l'AST resultant. | **Lèxic**<br>**Sintàctic** |
| 2 | Anàlisi de l'arbre AST resultant i construcció d'un gràfic que mostri les connexions entre els processos i les funcions del codi font, a més de mostrar altres parts clau. Aquest gràfic es pot generar proporcionant diverses opcions a l'usuari, i es fa gràcies al projecte de visualització de grafs **Graphviz**. | **Semàntic** |

En els següents apartats es mostrarà un exemple d'aplicació del projecte, les àrees que cobreix d'entre tot el llenguatge Erlang, l'organització de la pràctica, la compilació de la mateixa i, finalment, la seva execució.

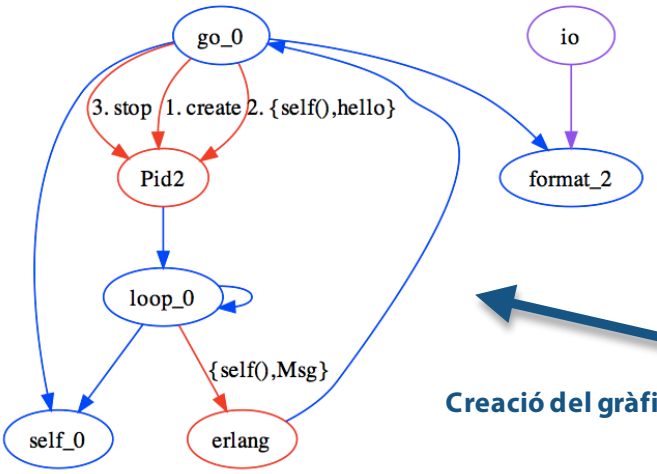
# 2 Exemple

Un exemple pràctic[1] d'aquest projecte seria:

```
go() ->
        Pid2 = spawn(echo, loop, []),
        Pid2 ! {self(), hello},
        receive
                {Pid2, Msg} ->
                        io:format("P1 ~w~n",[Msg])
        end,
        Pid2 ! stop.
loop() ->
        receive
                {From, Msg} ->
                        From ! {self(), Msg},
                        loop();
                stop ->
                        true
        end.
```

**Conversió a AST**

```
erlang
 | functions
 | | function
 | | | functionhead
 | | | | atom(go)
 | | | spawn
 | | | | var(Pid2)
 | | | | atom(echo)
 | | | | atom(loop)
 | | | | list
 | | | | send
 | | | | | var(Pid2)
 | | | | | envelope
 | | | | | | functionhead
 | | | | | | | atom(self)
 | | | | | | atom(hello)
 | | | | receive
 | | | | | received
 | | | | | | envelope
 | | | | | | | var(Pid2)
 | | | | | | | var(Msg)
 | | | | | | receivedfunctions
 | | | | | | | plainfunction
 | | | | | | | | functionhead
 | | | | | | | | | atom(io)
 | | | | | | | | | atom(format)
 | | | | | | | | | params
 | | | | | | | | | | list("P1 ~w~n")
 | | | | | | | | | | list
 | | | | | | | | | | | var(Msg)
 | | | | send
 | | | | | var(Pid2)
 | | | | | atom(stop)
 | | function
 | | | functionhead
 | | | | atom(loop)
 | | | receive
 | | | | received
 | | | | | envelope
 | | | | | | var(From)
 | | | | | | var(Msg)
 | | | | | receivedfunctions
 | | | | | | send
 | | | | | | | var(From)
 | | | | | | | envelope
 | | | | | | | | functionhead
 | | | | | | | | | atom(self)
 | | | | | | | | var(Msg)
 | | | | | | plainfunction
 | | | | | | | functionhead
 | | | | | | | | atom(loop)
 | | | | received
 | | | | | atom(stop)
 | | | | | receivedfunctions
 | | | | | | plainfunction
 | | | | | | | atom(true)
```




**Creació del gràfic**

[1] Inclòs com a **jp2** dins la carpeta de jocs de proves.

# 3 Àrees que cobreix

A continuació veurem quines àrees cobreix aquest analitzador d'entre tot Erlang.

- **Anàlisi lèxic**
  - Enters, ja sigui en base 10 (e.g. **-50**) o altres (e.g. **16#AB10F**)
  - Caràcters usats com a nombre (e.g. **$A**)
  - Coma flotant (e.g. **-12.34E-10**)
  - *Atoms*: cadenes que comencen per minúscula amb possibles barres baixes (e.g. **camp_de_futbol**) o cadenes entre cometes (e.g. **'Soc una persona'**)
  - Tuples, on el delimitador és la clau oberta (e.g. **{ 1, barcelona, cucut}** )
  - Llistes, on el delimitador és el claudàtor (e.g. **[ 123, segon_valor ]** )
  - Variables, que comencen amb majúscula (e.g. **Pid**)
- **Anàlisi sintàctic**
  - Combinacions de tuples, llistes i els altres tipus bàsics. E.g.:

    ```
    [ { persona, 'Leo Messi' }, { persona, 'Josep Guardiola'} ]
    ```

  - Funcions, de 0 a N paràmetres (e.g. **factorial(n)** )
  - Declaració de funcions sequencials, que poden tenir diversos casos. E.g.:

    ```
    factorial(0) -> 1 ;
    factorial(X) -> X * factorial(X - 1) .
    ```

  - Declaració de funcions concurrents: successions de creació de processos, enviament de missatges, i rebuda d'aquests. E.g.:

    ```
    go() ->
        Pid2 = spawn(echo, loop, []),
        Pid2 ! {self(), hello},
        receive
            {Pid2, Msg} ->
                io:format("P1 ~w~n",[Msg])
        end,
        Pid2 ! stop.
    ```

  - Declaració del mòdul. E.g.:

    ```
    -module(echo).
    ```

  - Exportació de funcions. E.g.:

**-export([go/0, invent/1]).**

- Importació de funcions. E.g.:

  **-import([g/1, h/1]).**

- Un programa en la seva totalitat.

- **Anàlisi semàntic**
  - Representació gràfica de
    - Mòdul del programa i mòduls de funcions importades
    - Funcions exportades
    - Funcions importades
    - Dependència entre funcions
    - Enviament de missatges entre processos

# 4 Organització de la pràctica

La pràctica està organitzada en directoris tal com segueix:

- **erlanger.pdf**: Aquest PDF explicatiu.
- **erlanger:** Directori on hi ha el codi font de la pràctica.
    - **erlanger.g:** Fitxer que s'encarrega de convertir el fitxer font en AST i enllaçar amb el generador del gràfic.
    - **graphgen.cc** i **graphgen.hh:** Fitxer font (i capçalera) que generen, a partir de l'arbre AST, les estructures de dades que el representen i que s'utilitzaran, després, per generar el gràfic.
    - **graphwriter.cc** i **graphwriter.hh:** Fitxer font (i capçalera) que generen, a partir d'unes estructures intermitges de dades, el gràfic final.
    - **Makefile:** Fitxer que permet generar l'executable **erlanger**.
    - **jp**: Directori on hi ha diversos jocs de proves.
    - **output:** Directori on es desa el gràfic, en el fitxer **graphic.pdf**.

# 5 Compilació

Per a compilar aquesta pràctica cal simplement:

- Complir aquests requisits
  - Entorn UNIX (no s'ha provat en entorn Windows)
  - Tenir instal·lat el compilador **g++**, així com **antlr** i **dlg** (de **pccts**)
  - Tenir instal·lat el **GraphViz** (i verificar que el programa **dot** està instal·lat)
- Seguir aquests instruccions
  - Extreure els fitxers presents a l'**erlanger.zip** lliurat en un directori qualsevol
  - Anar a aquell directori
  - Anar al directori **erlanger**
  - Compilar la pràctica (executar **make**)
- A continuació apareixeran, entre altres fitxers intermitjos, l'executable **erlanger**, del que parlarem en el següent apartat.

# 6 Execució i proves

Un cop tenim generat l'executable, vegem l'ús del mateix.

- Si fem **`./erlanger < codiFont`**, ho mostrarà tot.
- Si fem **`./erlanger -opcions < codiFont`**, mostrarà les parts que haguem assenyalat a ***opcions***:
  - **m** per mostrar mòduls
  - **e** per mostrar les funcions exportades
  - **i** per mostrar les funcions importades
  - **f** per mostrar les dependències entre funcions
  - **p** per mostrar processos
  - *Nota: Si escrivim **-meifp** ho mostrarà tot*
- Addicionalment, es pot indicar (opció **t**) que es mostri l'arbre del fitxer font.
- Per a executar un joc de proves determinat, simplement caldrà executar l'**erlanger** tal com s'acaba d'indicar amb qualsevol fitxer de la carpeta **jp** (e.g. **`./erlanger -fp jp/jp4`**)