P. Thiemann, M. Neubauer Wintersemester 2004/2005

Praktikum Compilerbau

`http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2004/`

Übungsblatt 4

Abgabe: 15.12.2004

Aufgabe 1 (Semantische Analyse):

Implementiere eine Programmtransformation, die die Javascript-Programme, die in dem unten markierten Fragment der abstrakten Syntax liegen, korrekt in die funktionale Zwischensprache (`common/Intermediate.ml`) übersetzt und alle andere Programme zurückweist. Als zusätzliche Vereinfachung dürfen Funktionen nur global definiert sein.

Lokal eingeführte Namen der Zwischensprache dürfen nur einmal gebunden werden. Der Gültigkeitsbereich einer lokal eingeführten Variablen erstreckt sich über den Rumpf der restlichen Anweisung. Der Gültigkeitsbereich von global definierten Funktionen erstreckt sich über den Rumpf aller anderen global definierten Funktionen sowie über die globale Einstiegs-Anweisung.

```
(* Identifier of variables and functions *)
type identifier = Identifier of string

(* Constant values *)
and constant =
    Number of float
  | String of string
  | True
  | False
  | This
  | Undefined
  | Null

and property_name =
    DynamicName of identifier
  | StaticName of constant

(* Assignment Operators *)
and assignment_operator =
    Regular_assign          (* = *)
  | Star_assign             (* *= *)
  | Slash_assign            (* /= *)
  | Rem_assign              (* %= *)
  | Plus_assign             (* += *)
  | Minus_assign            (* -= *)
  | Lshift_assign           (* <<= *)
  | Rsignedshift_assign     (* >>= *)
  | Runsignedshift_assign   (* >>>= *)
  | And_assign              (* &= *)
  | Xor_assign              (* ^= *)
  | Or_assign              (* |= *)

(* Unary Operators which cause side effects *)
and unary_operator =
    Incr_postfix        (* ++ *)
  | Decr_postfix        (* -- *)
  | Incr_prefix         (* ++ *)
  | Decr_prefix         (* -- *)
  | Delete              (* delete *)

(* Unary Operators which don't cause side effects *)
and sideeffect_free_unary_operator =
    Void                (* Void *)
  | Typeof              (* Typeof *)
  | Positive            (* + *)
  | Negative            (* - *)
  | Tilde               (* ~ *)
  | Bang                (* ! *)

(* Binary Operators *)
and binary_operator =
    Greater          (* > *)
  | Less             (* < *)
  | Eq               (* == *)
  | Le               (* <= *)
  | Ge               (* >= *)
  | Ne               (* != *)
  | Eqq              (* === *)
  | Neq              (* !== *)
  | Sc_or            (* || *)
  | Sc_and           (* && *)
  | Plus             (* + *)
  | Minus            (* - *)
  | Star             (* * *)
  | Slash            (* / *)
  | Bit_and          (* & *)
  | Bit_or           (* | *)
  | Xor              (* ^ *)
  | Rem              (* % *)
  | Lshift           (* << *)
  | Rsignedshift     (* >> *)
  | Runsignedshift   (* >>> *)
  | Instanceof       (* instanceof *)
  | In               (* in *)

(* Possibilities of the definition part in a for-loop *)
and for_bracket =
    Regular of expression option * expression option * expression option
  | Regular_var of
      (identifier * expression option) list * expression option *
```

```
        expression option
  | With_in of expression * expression
  | With_in_and_var of (identifier * expression option) * expression

(* Expressions *)
and expression =
    Constant of constant
  | Variable of identifier
  | Sequence of expression list
  | Array_construction of (expression option) list
  | Array_access of expression * expression
  | Object_construction of (property_name * expression) list
  | Object_access of expression * identifier
  | Function_expression of
      identifier option * identifier list * source_element list
  | Assign of expression * assignment_operator * expression
  | Unop_without_sideeffect of expression * sideeffect_free_unary_operator
  | Unop of expression * unary_operator
  | Binop of expression * binary_operator * expression
    (* cond ? e1 : e2  - Syntax *)
  | Conditional of expression * expression * expression
  | Function_call of expression * expression list
  | Method_call of expression * identifier * expression list
  | New_construct of expression * expression list
  | New_expression of expression
  | RegExp of string * string

(* Statements *)
and statement =
    Skip
  | Block of statement list
  | Variable_declaration of (identifier * expression option) list
  | Expression of expression
  | If of expression * statement * statement option
  | Do of statement * expression
  | While of expression * statement
  | For of for_bracket * statement
  | Continue of identifier option
  | Break of identifier option
  | Return of expression option
  | With of expression * statement
  | Labelled_statement of identifier * statement
  | Switch of
      expression *
        (* regular case-cases *)
        (expression * statement list option) list *
        (* default-case and content optional *)
        statement list option option *
        (* additional case-cases after default *)
        (expression * statement list option) list
  | Throw of expression
    (* try catch do finally *)
  | Try_catch_finally of
      statement * (identifier * statement) option * statement option

and source_element =
     Statement of statement
   | Function_declaration of
       identifier option * identifier list * source_element list

and program = Program of source_element list
```