PADS/Haskell Grammar

Kathleen Fisher

October 5, 2014

$\langle \mathit{decls} \rangle ::= \langle \mathit{decl} \rangle^*$

$\langle \mathit{dec} \rangle ::= \langle \mathit{typeDecl} \rangle \mid \langle \mathit{dataDecl} \rangle \mid \langle \mathit{newDecl} \rangle \mid \langle \mathit{obtainDecl} \rangle$

$\langle \mathit{typeDecl} \rangle ::= \texttt{type}\ \langle \mathit{padsID} \rangle\ \langle \mathit{haskell\text{-}pat} \rangle?\ \texttt{=}\ \langle \mathit{ptype} \rangle$

$\langle \mathit{dataDecl} \rangle ::= \texttt{data}\ \langle \mathit{padsID} \rangle\ \langle \mathit{haskell\text{-}pat} \rangle?\ \texttt{=}\ \langle \mathit{dataRHS} \rangle\ \langle \mathit{derives} \rangle?$

$\langle \mathit{newDecl} \rangle ::= \texttt{newtype}\ \langle \mathit{padsID} \rangle\ \langle \mathit{haskell\text{-}pat} \rangle?\ \texttt{=}\ \langle \mathit{newRHS} \rangle\ \langle \mathit{derives} \rangle?$

$\langle \mathit{obtainDecl} \rangle ::= \texttt{obtain}\ \langle \mathit{padsID} \rangle\ \texttt{from}\ \langle \mathit{ptype} \rangle\ \texttt{using}\ \langle \mathit{expression} \rangle$

$\langle \mathit{padsID} \rangle ::= \langle \mathit{upper} \rangle \langle \mathit{lower} \rangle^*$

$\langle \mathit{ptype} \rangle ::= \langle \mathit{constrain} \rangle \mid \langle \mathit{obtain} \rangle \mid \langle \mathit{partition} \rangle \mid \langle \mathit{listTy} \rangle \mid \langle \mathit{value} \rangle \mid \langle \mathit{btype} \rangle$

$\langle \mathit{constrain} \rangle ::= \texttt{constrain}\ \langle \mathit{haskell\text{-}pat} \rangle\ \texttt{::}\ \langle \mathit{ptype} \rangle\ \langle \mathit{predic} \rangle$

$\langle \mathit{obtain} \rangle ::= \texttt{obtain}\ \langle \mathit{ptype} \rangle\ \texttt{from}\ \langle \mathit{ptype} \rangle\ \texttt{using}\ \langle \mathit{expression} \rangle$

$\langle \mathit{partition} \rangle ::= \texttt{partition}\ \langle \mathit{ptype} \rangle\ \texttt{using}\ \langle \mathit{expression} \rangle$

$\langle \mathit{listTy} \rangle ::= \texttt{[}\ \langle \mathit{listInside} \rangle\ \texttt{]}\ \langle \mathit{listEnd} \rangle$

$\langle \mathit{listInside} \rangle ::= \langle \mathit{ptype} \rangle\ (\texttt{|}\ \langle \mathit{ptype} \rangle)?$

$\langle \mathit{listEnd} \rangle ::= \texttt{terminator}\ \langle \mathit{ptype} \rangle \mid \texttt{length}\ \langle \mathit{expression} \rangle$

$\langle \mathit{value} \rangle ::= \texttt{value}\ \langle \mathit{expression} \rangle\ \texttt{::}\ \langle \mathit{ptype} \rangle$

$\langle \mathit{btype} \rangle ::= \langle \mathit{etype} \rangle\ \langle \mathit{atype} \rangle^*\ \langle \mathit{expression} \rangle?$

$\langle \mathit{etype} \rangle ::= \langle \mathit{atype} \rangle \mid \langle \mathit{expression} \rangle$

$\langle \mathit{atype} \rangle ::= \langle \mathit{tuple} \rangle \mid \texttt{[}\ \langle \mathit{listInside} \rangle\ \texttt{]} \mid \langle \mathit{qualUpper} \rangle \mid \langle \mathit{tyvar} \rangle$

$\langle \mathit{tuple} \rangle ::= \texttt{(}\ ((\langle \mathit{ptype} \rangle\ (\texttt{,}\ \langle \mathit{ptype} \rangle)^*)?\ \texttt{)}$

$\langle \mathit{dataRHS} \rangle ::= \langle \mathit{switchTy} \rangle \mid \langle \mathit{constrs} \rangle$

$\langle \mathit{switchTy} \rangle ::= \texttt{case}\ \langle \mathit{expression} \rangle\ \texttt{of}\ \langle \mathit{branch} \rangle\ (\texttt{|}\ \langle \mathit{branch} \rangle)^*$

⟨_branch_⟩ ::= ⟨_haskell-pat_⟩ `->` ⟨_constr_⟩

⟨_constrs_⟩ ::= ⟨_constr_⟩ (`|` ⟨_constr_⟩)*

⟨_constr_⟩ ::= ⟨_upper_⟩ ⟨_record_⟩ ⟨_predic_⟩? | ⟨_upper_⟩ ⟨_constrArgs_⟩? ⟨_predic_⟩?

⟨_constrArgs_⟩ ::= (`!`? ⟨_etype_⟩)+

⟨_record_⟩ ::= `{` ⟨_field_⟩ (`,` ⟨_field_⟩)* `}`

⟨_field_⟩ ::= ⟨_lower_⟩ `::` ⟨_ftype_⟩ ⟨_predic_⟩?
| ⟨_lower_⟩ `= value` ⟨_expression_⟩ `::` ⟨_ftype_⟩ ⟨_predic_⟩?
| ⟨_ftype_⟩ ⟨_predic_⟩?

⟨_ftype_⟩ ::= `!` ⟨_atype_⟩ | ⟨_ptype_⟩

⟨_newRHS_⟩ ::= ⟨_upper_⟩ ⟨_record1_⟩ ⟨_predic_⟩? | ⟨_upper_⟩ ⟨_atype_⟩ ⟨_predic_⟩?

⟨_record1_⟩ ::= `{` (⟨_ftype_⟩ `,`)* ⟨_field1_⟩ (`,` ⟨_ftype_⟩)* `}`

⟨_field1_⟩ ::= ⟨_lower_⟩ `::` ⟨_ptype_⟩ ⟨_predic_⟩?

⟨_literal_⟩ ::= ⟨_charLit_⟩ | ⟨_reLit_⟩ | ⟨_stringLit_⟩ | ⟨_intLit_⟩ | ⟨_qualLower_⟩ | ⟨_qualUpper_⟩

⟨_predic_⟩ ::= `where` ⟨_expression_⟩

⟨_expression_⟩ ::= ⟨_h-exp_⟩ | ⟨_literal_⟩

⟨_h-exp_⟩ ::= `<|` ⟨_haskell-exp_⟩ `|>`

⟨_derives_⟩ ::= `deriving` ⟨_qualUpper_⟩ | `deriving (` ⟨_qualUppers_⟩ `)`

⟨_tyvar_⟩ ::= ⟨_lower_⟩

⟨_qualUpper_⟩ ::= ⟨_upper_⟩ | ⟨_qualUpper_⟩ `.` ⟨_upper_⟩

⟨_qualLower_⟩ ::= ⟨_lower_⟩ | ⟨_qualUpper_⟩ `.` ⟨_lower_⟩

⟨_qualUppers_⟩ ::= ⟨_qualUpper_⟩ | ⟨_qualUpper_⟩ `,` ⟨_qualUppers_⟩

⟨_haskell-pat_⟩ ::= Parsed according to Language.Haskell.Meta.parsePat

⟨_haskell-exp_⟩ ::= Parsed according to Language.Haskell.Meta.pareExp

⟨_upper_⟩ ::= ⟨_identifier_⟩ with capitalized first character

⟨_lower_⟩ ::= ⟨_identifier_⟩ with lowercase first character

⟨_reLit_⟩ ::= Anything contained within single quotes (')

⟨_charLit_⟩ ::= As defined in Text.Parsec.Token

⟨_stringLit_⟩ ::= As defined in Text.Parsec.Token

$\langle \textit{intLit} \rangle$ ::= As defined in Text.Parsec.Token

$\langle \textit{identifier} \rangle$ ::= As defined in Text.Parsec.Token

$\langle \textit{whiteSpace} \rangle$ :: As defined in Text.Parsec.Token