Syntax for Handy-LK

(short `hlk`-files)

Hendrik Spohr

December 7, 2006

IMPORTANT: This is a draft!
The grammar may change slightly in the future.

1 HLK Grammar

2 HLK File

⟨*hlk-file*⟩ ::= (⟨*hlk-statement*⟩ `;` | ⟨*single-line-comment*⟩)*

⟨*single-line-comment*⟩ ::= `#` (⟨*any-character*⟩)* ⟨*new-line*⟩

⟨*hlk-statement*⟩ ::= ⟨*reference*⟩
| ⟨*definition*⟩

⟨*reference*⟩ ::= `reference` ⟨*file*⟩

⟨*definition*⟩ ::= ⟨*type-definition*⟩
| ⟨*variable-definition*⟩
| ⟨*constant-definition*⟩
| ⟨*function-definition*⟩
| ⟨*predicate-definition*⟩
| ⟨*axiom-definition*⟩
| ⟨*equality-definition*⟩
| ⟨*proof-definition*⟩

3 Types

⟨*type-definition*⟩ ::= `define type` ⟨*identifier-list*⟩
[`fresh variable starts with` ⟨*identifier*⟩]

4 Terms

⟨*variable-definition*⟩ ::= `define variable` ⟨*identifier-list*⟩ `of type` ⟨*type*⟩

⟨*constant-definition*⟩ ::= `define constant` ⟨*identifier-list*⟩ `of type` ⟨*type*⟩

⟨*function-definition*⟩ ::= ⟨*atom-funcdef*⟩
| ⟨*funcdef-by-term*⟩
| ⟨*funcdef-by-formula*⟩
| ⟨*recursive-funcdef*⟩

⟨*atom-funcdef*⟩ ::= `define` [⟨*notation-clause*⟩] `function` ⟨*identifier-list*⟩
`of type` ⟨*type-list*⟩ `to` ⟨*type*⟩ [⟨*weight-clause*⟩]
| `define function` ⟨*func-prototype*⟩
`to` ⟨*type*⟩ [⟨*weight-clause*⟩]

⟨*funcdef-by-term*⟩ ::= `define` [⟨*notation-clause*⟩] `function` (⟨*identifier*⟩ |
⟨*func-prototype*⟩) `by` ⟨*term*⟩ [⟨*weight-clause*⟩]

⟨*funcdef-by-formula*⟩ ::= `define` [⟨*notation-clause*⟩] `function` ⟨*identifier*⟩
`through` ⟨*formula*⟩ `is defined by` ⟨*formula*⟩
[⟨*weight-clause*⟩]

⟨*recursive-funcdef*⟩ ::= `define recursive function` ⟨*identifier*⟩
[`(` ⟨*variable-list*⟩ `)`] `to` ⟨*type*⟩
(`at level` ⟨*rec-level-term*⟩ `by` ⟨*term*⟩)+

⟨*func-prototype*⟩ ::= ⟨*identifier*⟩ `(` ⟨*variable-list*⟩ `)`
| `(` ⟨*variable*⟩ ⟨*identifier*⟩ ⟨*variable*⟩ `)`

5 Predicates

⟨*predicate-definition*⟩ ::= ⟨*atom-preddef*⟩
| ⟨*preddef-by-formula*⟩
| ⟨*recursive-preddef*⟩

⟨*atom-preddef*⟩ ::= `define` [⟨*notation-clause*⟩] `atom predicate`
⟨*identifier-list*⟩ [`of type` ⟨*type-list*⟩]

⟨*preddef-by-formula*⟩ ::= `define` [⟨*notation-clause*⟩] `predicate` ⟨*identifier*⟩
[`(` ⟨*variable-list*⟩ `)`] `by` ⟨*formula*⟩

⟨*recursive-preddef*⟩ ::= `define recursive predicate` ⟨*identifier*⟩
[`(` ⟨*variable-list*⟩ `)`]
(`at level` ⟨*rec-level-term*⟩ `by` ⟨*formula*⟩)+

⟨*axiom-definition*⟩ ::= `define axiom` ⟨*sequent*⟩ [`named as` ⟨*identifier*⟩]

⟨*equality-definition*⟩ ::= `define equality for` ⟨*variable*⟩ `=` ⟨*variable*⟩
`as` ⟨*formula*⟩

⟨*weight-clause*⟩ ::= `with weight` ⟨*weight*⟩

⟨*notation-clause*⟩ ::= `prefix`
| `infix`

6 Proof Definition

⟨*proof-definition*⟩ ::= `define proof` ⟨*identifier*⟩ (⟨*proof-meta-decl*⟩)*
⟨*proof-body*⟩
| `define recursive proof` ⟨*identifier*⟩
(⟨*proof-meta-decl*⟩)*
(`at level` ⟨*rec-level-term*⟩ ⟨*proof-body*⟩)+

⟨*proof-meta-decl*⟩ ::= `with meta` [⟨*notation-clause*⟩]
`formula` ⟨*identifier-list*⟩ [`of type` ⟨*type-list*⟩] `;`
| `with meta term` ⟨*identifier-list*⟩ `of type` ⟨*type*⟩ `;`
| `with meta` [⟨*notation-clause*⟩]
`term` ⟨*identifier-list*⟩ `of type` ⟨*type-list*⟩ `to` ⟨*type*⟩
[⟨*weight-clause*⟩] `;`

⟨*proof-body*⟩ ::= `proves` ⟨*sequent*⟩ `;`
(⟨*proof-step*⟩ `;`)+
[⟨*proof-end-step*⟩ `;`]

⟨*proof-step*⟩ ::= `with` ⟨*unary-rule*⟩ ⟨*sequent*⟩
| `with cut` ⟨*formula*⟩
[`left` | `right`] ⟨*proof-ref-clause*⟩
| `with` ⟨*binary-rule*⟩ ⟨*sequent*⟩
[`left` | `right`] ⟨*proof-ref-clause*⟩
| `with paramod by` ⟨*equation*⟩ `right` ⟨*sequent*⟩

⟨*proof-end-step*⟩ ::= `with cut` ⟨*formula*⟩
`left` ⟨*proof-ref-clause*⟩
`right` ⟨*proof-ref-clause*⟩
| `with` ⟨*binary-rule*⟩
`left` ⟨*proof-ref-clause*⟩

right ⟨*proof-ref-clause*⟩
| continued ⟨*proof-ref-clause*⟩
| explicit axiom ⟨*sequent*⟩

⟨*unary-rule*⟩ ::= not left
| not right
| and left
| or right
| impl right
| all left
| all right
| ex left
| ex right
| undef ⟨*identifier*⟩
| undef equality of type ⟨*type*⟩ , ⟨*type*⟩
| auto propositional

⟨*binary-rule*⟩ ::= and right
| or left
| impl left
| paramod

⟨*proof-ref-clause*⟩ ::= by proof ⟨*proof*⟩ [(⟨*proof-arg-list*⟩)]
| auto propositional
| explicit axiom ⟨*sequent*⟩

⟨*proof-arg-list*⟩ ::= ⟨*proof-arg*⟩
| ⟨*proof-arg*⟩ , ⟨*proof-arg-list*⟩

⟨*proof-arg*⟩ ::= ⟨*formula*⟩
| ⟨*term*⟩

7 Sequent and Formulas

⟨*sequent*⟩ ::= [⟨*formula-list*⟩] :- [⟨*formula-list*⟩]

⟨*formula-list*⟩ ::= ⟨*formula*⟩
| ⟨*formula*⟩ , ⟨*formula-list*⟩

⟨*formula*⟩ ::= ⟨*subformula*⟩
| ⟨*term*⟩ ⟨*infix-predicate-symbol*⟩ ⟨*term*⟩
| ⟨*sub-formula*⟩ and ⟨*sub-formula*⟩
| ⟨*sub-formula*⟩ or ⟨*sub-formula*⟩
| ⟨*sub-formula*⟩ impl ⟨*sub-formula*⟩

⟨*sub-formula*⟩ ::= `(` ⟨*formula*⟩ `)`
| `not` ⟨*sub-formula*⟩
| `all` ⟨*variable*⟩ ⟨*sub-formula*⟩
| `ex` ⟨*variable*⟩ ⟨*sub-formula*⟩
| ⟨*predicate*⟩ [`(` ⟨*term-list*⟩ `)`]

8 Terms

⟨*term*⟩ ::= ⟨*sub-term*⟩
| ⟨*sub-term*⟩ ⟨*infix-function-symbol*⟩ ⟨*term*⟩

⟨*sub-term*⟩ ::= ⟨*variable*⟩
| ⟨*constant*⟩
| ⟨*function*⟩ `(` ⟨*term-list*⟩ `)`

9 Symbols

⟨*variable*⟩ ::= ⟨*identifier*⟩

⟨*variable-list*⟩ ::= ⟨*variable*⟩ (`,` ⟨*variable*⟩)*

⟨*constant*⟩ ::= ⟨*identifier*⟩

⟨*infix-function-symbol*⟩ ::= ⟨*identifier*⟩

⟨*function*⟩ ::= ⟨*identifier*⟩
| ⟨*identifier*⟩ `[` ⟨*recursion-expr*⟩ `]`

⟨*infix-predicate-symbol*⟩ ::= ⟨*identifier*⟩

⟨*predicate*⟩ ::= ⟨*identifier*⟩
| ⟨*identifier*⟩ `[` ⟨*recursion-expr*⟩ `]`

⟨*recursion-variable*⟩ ::= ⟨*identifier*⟩

10 Recursion Expressions

⟨*recursion-expr*⟩ ::= ⟨*positive-integer*⟩
| ⟨*recursion-variable*⟩
| ⟨*recursion-variable*⟩
⟨*recursion-expr-op*⟩ ⟨*positive-integer*⟩

⟨*recursion-expr-op*⟩ ::= `+` | `-` | `*` | `/`

⟨*rec-level-term*⟩ ::= ⟨*positive-integer*⟩
| ⟨*recursion-variable*⟩
| ⟨*recursion-variable*⟩ `+` ⟨*positive-integer*⟩
| ⟨*recursion-variable*⟩ `*` ⟨*positive-integer*⟩