1 Nevek

- alfanumerikus, azaz betûk, számjegyek, percjelek és aláhúzás-jelek olyan sorozata, amely betûvel vagy percjellel kezdõdik;
- szimbolikus, azaz az alábbi jelek tetszõleges, nem üres sorozata:

```
! % & $ # + - / : < = > ? @ \ ~ ‘ ^ | *
```

Névként nem használhatók a következõ, ún. fenntartott szavak vagy kulcsszók:

```
abstype and andalso as case do datatype else end
exception fn fun handle if in infix infixr let local
nonfix of op open orelse raise rec sig signature
struct structure then type val with withtype while
( ) [ ] { } , : :> ; ... _ | = => -> #
```

A neveket különféle osztályokba soroljuk:

var	értékváltozó	value variable	long
con	értékkonstruktor	value constructor	long
excon	kivételkonstruktor	exception constructor	long
tyvar	típusváltozó	type variable	
tycon	típuskonstruktor	type constructor	long
lab	mezõnév	record label	
unitid	modulnév	unit identifier	

- A típusváltozó olyan alfanumerikus név, amely percjellel kezdõdik, pl. `'a`.
- A mezõnév tetszõleges név lehet, vagy olyan pozitív egész , amely nem 0-val kezdõdik.
- Minden 'long' jelzésû X osztálynak van egy *longX* párja. A *longX* osztályba tartozó nevek rövid és hosszú (ún. minõsített) alakban is felírhatók. A *rövid alak* csak egy névbõl, a *hosszú alak* egy modulnévbõl, egy pontból és egy névbõl áll:

longx	::=	x	név	identifier
		unitid. x	minõsített név	qualified identifier

2 Infix operátorok

Egy név az `infix` vagy az `infixr` direktívával *infix* helyzetûnek deklarálható. Ha az *id* név infix helyzetû, akkor az exp_1 *id* exp_2 kifejezés, szükség esetén zárójelek között, minden olyan helyen használható, ahol az *id*(exp_1, exp_2) vagy

az *id*{1=exp_1, 2=exp_2} kifejezések egyébként használhatók. Infix helyzetû nevek mintában szintén használhatók.
Egy minõsített nevet, vagy egy olyan nevet, amelyet az `op` szócska elõz meg, sohasem lehet infix helyzetben alkalmazni. Az `infix`, `infixr` és `nonfix` direktívák szintaxisa a következõ ($n \geq 1$):

`infix`	$\langle$`d`$\rangle$	id_1 ... id_n	balra köt	left associative
`infixr`	$\langle$`d`$\rangle$	id_1 ... id_n	jobbra köt	right associative
`nonfix`		id_1 ... id_n	nem köt	non-associative

A `d` decimális számjegy opcionális, és a nevek precedenciáját adja meg; alapértelmezés szerinti értéke 0. Nagyobb szám magasabb precedenciát jelent. Az `infix`, `infixr` és `nonfix` direktívák érvényességi tartománya a szokásos, azaz a `let` kifejezésekben és a `local` deklarációkban deklarált nevek láthatósági szabályai változatlanok.
Azonos precedenciájú, de különféle balra kötõ operátorok balra, azonos precedenciájú, de különféle jobbra kötõ operátorok pedig jobbra kötnek. Tilos viszont azonos precendeciájú, de különbözõ kötésû operátorokat használni ugyanabban a kifejezésben.

3 Általánosan használt jelölések a nyelvtanban

- Minden szintaktikai osztályt változatok listájaként adunk meg, mégpedig soronként egy változatot. Üres sor üres kifejezést jelent.
- A $<$ és a $>$ csúcsos zárójelpárok opcionális kifejezést fognak közre.
- The brackets and enclose optional phrases.
- Tetszõleges X szintaktikai osztályra (amelyre az x értelmezve van) az alábbiak szerint definiáljuk az *Xseq* szintaktikai osztályt (amelyre az *xseq* értelmezve van):

xseq	::=	x	egyelemû sorozat	singleton sequence
			üres sorozat	empty sequence
		x_1, ..., x_n	sorozat	sequence, $n \geq 1$

- A kifejezésváltozatokat csökkenõ precendenciájuk sorrendjében adjuk meg.
- Az L és az R betûk a balra (L), ill. a jobbra (R) kötést jelzik.
- A típusok szintaxisa erõsebben köt, mint a kifejezéseké.
- Minden ismétlõdõ konstrukció (pl. a *klózsorozat)* a lehetõ legtovább terjeszkedik jobbra. Ezért egy `case`-kifejezést egy másik `case-` vagy egy `fn`-kifejezésen, vagy egy `fun`-definíción belül esetleg zárójelek közé kell rakni.

4 A Standard ML nyelvtana (Core language)

4.1 Kifejezések és klózsorozatok

exp	::=	*infexp*		
		exp : *ty*	típusmegkötés (L)	type constraint (L)
		exp1 `andalso` *exp2*	szekvenciális konjunkció	sequential conjunction
		exp1 `orelse` *exp2*	szekvenciális alternáció	sequential disjunction
		exp `handle` *match*	kivétel kezelése	handle exception
		`raise` *exp*	kivétel jelzése	raise exception
		`if` *exp1* `then` *exp2* `else` *exp3*	feltételes kifejezés	conditional expression
		`while` *exp1* `do` *exp2*	iteráció	iteration
		`case` *exp* `of` *match*	esetszétválasztás	case analysis
		`fn` *match*	lambda-kifejezés	function expression

infexp	::=	*appexp*		
		infexp1 id infexp2	infix alkalmazás	infixed application

appexp	::=	*atexp*		
		appexp atexp	(prefix) alkalmazás	application

atexp	::=	*scon*	különleges állandó	special constant (Sec. 3)
		$<$`op`$>$ *longvar*	értékváltozó	value variable
		$<$`op`$>$ *longcon*	adatkonstruktor	value constructor
		$<$`op`$>$ longexcon	kivételkonstruktor	exception constructor
		`{` $<$ *exprow* $>$ `}`	rekord	record
		# *lab*	rekordszelektor	record selector
		()	nullas	0-tuple
		`(`exp_1`,` ...`,` exp_n`)`	ennes	n-tuple, $n \geq 2$
		`[`exp_1`,` ...`,` exp_n`]`	lista	list, $n \geq 0$
		`#[`exp_1`,` ...`,` exp_n`]`	vektor	vector, $n \geq 0$
		`(`exp_1`;` ...`;` exp_n`)`	kifejezéssorozat	sequence, $n \geq 2$
		`let` *dec* `in` exp_1`;` ...`;` exp_n `end`	lokális kifejezés	local expression, $n \geq 1$
		`(` *exp* `)`		

exprow	::=	*lab* = *exp* $<$, *exprow* $>$	kifejezéssor	expression row

match	::=	*mrule* $<$ | *match* $>$	klózsorozat, változatsorozat	match

mrule	::=	*pat* `=>` *exp*	klóz, változat	match rule

4.2 Declarációk és kötések

dec	::=	`val` *tyvarseq valbind*	értékdeklaráció	value declaration
		`fun` *tyvarseq fvalbind*	függvénydeklaráció	function declaration
		`type` *typbind*	típusdeklaráció	type declaration
		`datatype` *datbind* $<$ `withtype` *typbind* $>$	datatype-deklaráció	datatype declaration
		`abstype` *datbind* $<$ `withtype` *typbind* $>$ `with` *dec* `end`	abstype-deklaráció	abstype declaration
		`exception` *exbind*	kivételdeklaráció	exception declaration
		`local` dec_1 `in` dec_2 `end`	lokális deklaráció	local declaration
		`open` $unitid_1$ ... $unitid_n$	open-deklaráció	open declaration, $n \geq 1$
			üres deklaráció	empty declaration
		dec_1 $<$`;`$>$ dec_2	deklaráció-sorozat	sequential declaration
		`infix` $<$*d*$>$ id_1 id_n	infix-direktíva	infix (left) directive, $n \geq 1$
		`infixr` $<$*d*$>$ id_1 id_n	infixr-direktíva	infix (right) directive, $n \geq 1$
		`nonfix` id_1 ... id_n	nonfix-direktíva	nonfix directive, $n \geq 1$

valbind	::=	*pat* = *exp* $<$ `and` *valbind* $>$	értékkötés	value binding
		`rec` *valbind*	rekurzív kötés	recursive binding

fvalbind	::=	$<$`op`$>$ *var* $atpat_{11}$ ... $atpat_{1n}$ $<$`:`*ty*$>$ = exp_1	$m, n \geq 1$
		| $<$`op`$>$ *var* $atpat_{21}$ ... $atpat_{2n}$ $<$`:`*ty*$>$ = exp_2	
		| ...	
		| $<$`op`$>$ *var* $atpat_{m1}$ ... $atpat_{mn}$ $<$`:`*ty*$>$ = exp_m	
		$<$ `and` *fvalbind* $>$	

typbind	::=	*tyvarseq tycon* = *ty* < `and` *typbind* >

datbind	::=	*tyvarseq tycon* = *conbind* < `and` *datbind* >

conbind	::=	<`op`> *con* <`of` *ty*> < `|` *conbind* >

exbind	::=	<`op`> *excon* <`of` *ty*> < `and` *exbind* >
		<`op`> *excon* = `op` *longexcon* < `and` *exbind* >

Megjegyzés: *fvalbind* fenti definíciójában, ha *var* infix helyzetûnek van deklarálva, akkor vagy meg kell elõznie az `op` szócskának, vagy infix helyzetben kell használni. Ez azt jelenti, hogy a klózok elején `op` *var* `(`*atpat*`,` *atpat'*`)` helyett `(`*atpat var atpat'*`)` írható. A zárójelek elhagyhatók, ha *atpat'* után közvetlenül `:`*ty* vagy `=` áll.

4.3 Típuskifejezések

ty	::=	*tyvar*	típusváltozó	type variable
		`{` < *tyrow* > `}`	rekordtípus	record type expression
		tyseq longtycon	típuskonstrukció	type construction
		ty_1 `*` ... `*` ty_n	ennes típus	tuple type, $n \geq 2$
		ty_1 `->` ty_2	függvénytípus	function type expression
		`(` *ty* `)`		

tyrow	::=	*lab* `:` *ty* < `,` *tyrow* >	típuskifejezés-sor	type-expression row

4.4 Minták

atpat	::=	`_`	mindenesjel	wildcard
		scon	különleges állandó	special constant (Sec. 3)
		<`op`> *var*	változó	variable
		<`op`> *longcon*	konstruktor	constructor
		<`op`> *longexcon*	kivételkonstruktor	exception constructor
		`{` < *patrow* > `}`	rekord	record
		`()`	nullas	0-tuple
		`(`pat_1`,` ..., pat_n`)`	ennes	n-tuple, $n \geq 2$
		`[`pat_1`,` ..., pat_n`]`	lista	list, $n \geq 0$
		`#[`pat_1`,` ..., pat_n`]`	vektor	vector, $n \geq 0$
		`(` *pat* `)`		

patrow	::=	`...`	mindenesjel	wildcard
		lab = *pat* < `,` *patrow* >	mintasor	pattern row
		lab <`:`*ty*> < `as` *pat* > < `,` *patrow* >	címke mint változó	label as variable

pat	::=	*atpat*	atomi minta	atomic pattern
		<`op`> *longcon atpat*	értékkonstrukció	value construction
		<`op`> *longexcon atpat*	kivételkonstrukció	exception construction
		pat_1 *con* pat_2	infix értékkonstrukció	infixed value construction
		pat_1 *excon* pat_2	infix kivételkonstrukció	infixed exception construction
		pat `:` *ty*	minta típusmegkötéssel	typed pattern
		<`op`> *var* <`:`*ty*> `as` *pat*	réteges minta	layered pattern

4.5 Szintaktikai korlátozások

- Ugyanarra a névre *(var)* kétszer nem illesztheto minta. Ugyanarra a mezõnévre *(lab)* kétszer nem illeszthetõ kifejezéssor, mintasor vagy típussor.
- Ugyanaz a név nem köthetõ le kétféleképpen egy *valbind, typbind, datbind* vagy *exbind* deklarációban. A *datbind* deklarációban ugyanez érvényes az adatkonstruktorokra is.
- Ugyanaz a típusváltozó *(tyvar)* nem szerepelhet kétszer egy *tyvarseq* sorozatban valamely *typbind* vagy *datbind* deklaráció bal oldali *tyvarseq tycon* részében. Minden olyan típusváltozónak *(tyvar),* amelyik elõfordul a jobb oldalon, szerepelnie kell *tyvarseq*-ben.
- A `rec`-et követõ minden *pat* = *exp* értékkötésben az *exp*-nek, szükség esetén zárójelek között, `fn` *match* alakúnak kell lennie, ahol egy vagy több névhez típusmegkötés is társítható.
- `true`, `false`, `nil`, `::` és `ref`nem kaphat új értéket egy *valbind, datbind* vagy *exbind* deklarációban. Az `it` név nem kaphat új értéket egy *datbind* vagy *exbind* deklarációban.