t	::=		terms	S	::=		tags
		b	boolean value			Int	integer tag
		n	numeric value			Bool	boolean tag
		op	operator			Fun	function tag
		$\lambda x.\ t$	abstraction				
		x	variable	v	::=		configuration − values
		$t\ t$	application			b	boolean value
		mlet $x = t$ in t	overloading let			n	numeric value
		$t :: T$	ascription			op	operator
						$(\lambda x.\ t)[s]$	closure
b	::=		boolean value				
		true	true value	c	::=		configurations
		false	false value			v	
						$t[s]$	
op	::=		operators			$c\ c$	
		add1	sum			mlet $x = c$ in c	
		not	negation			$c :: T$	
						error	
T	::=		types	s	::=		explicit substitutions
		Int	type of integers			$\bullet$	empty substitution
		Bool	type of booleans			$x \mapsto \{(\overline{v : S})\}, s$	variable substitution
		$T \to T$	type of functions				

Figure 1: Syntax of the simply typed lambda-calculus vith overloading.

- Non deterministic.
- Type error detection.
- Dispatch error detection. In the case of the lambda functions, it is not effective because of the environment contains at least a value with tag function, it is not detected dispatch error. let, ascription.
- Without type annotation in lambda functions or mlet, only in ascription.
- Semantic "tag driven", introducing flat tag in the environment.

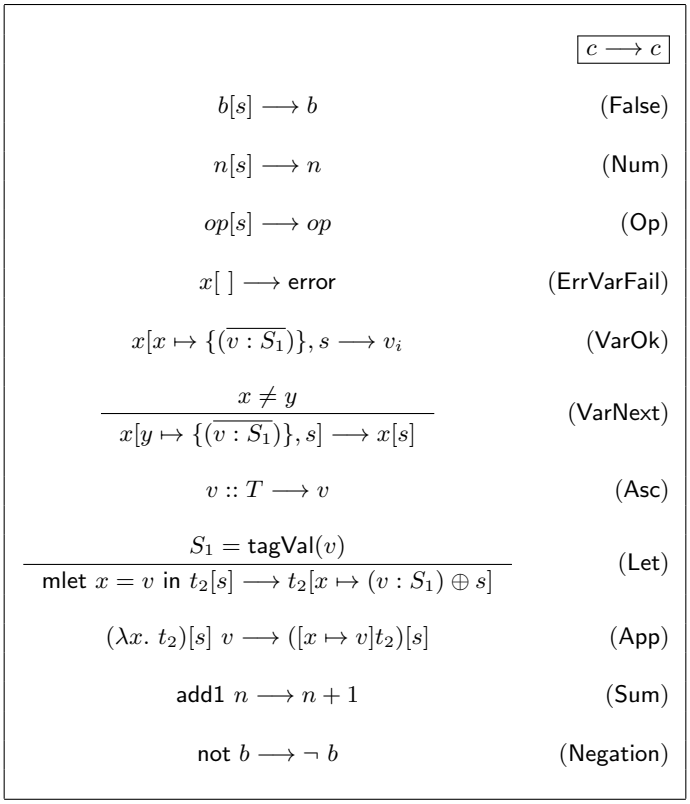

Figure 2: Configuration reduction rules.

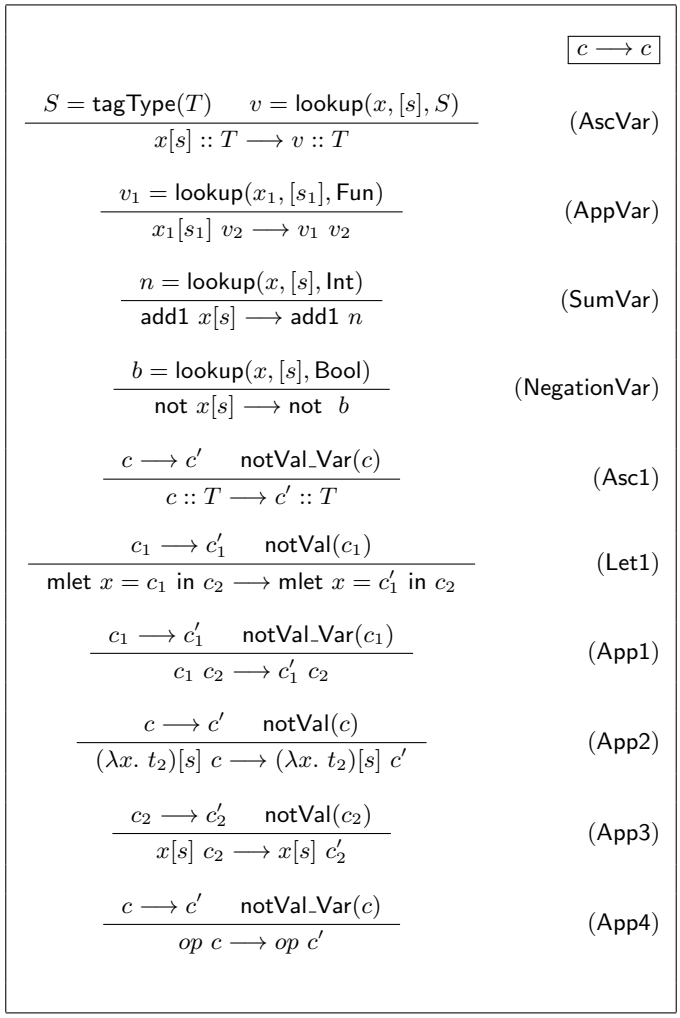

Figure 3: Configuration reduction rules.