

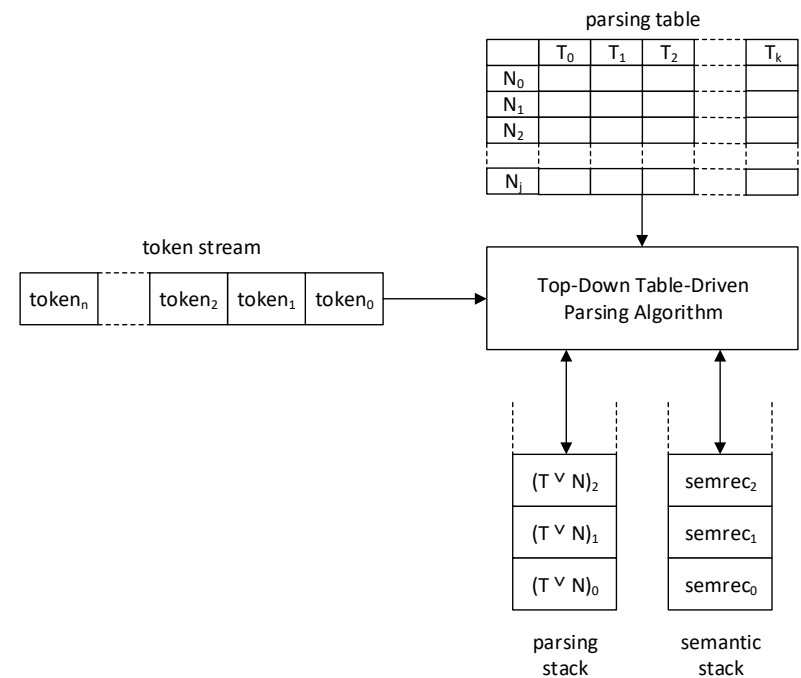
COMPILER DESIGN

Table-driven syntax-directed translation

Top-down table-driven syntax-directed translation

Top-down table-driven syntax-directed translation

- Augment the parser algorithm to implement attribute migration.
 - Introduce additional symbols in the grammar's right hand sides for **semantic actions** that process **semantic attributes**.
 - The grammar becomes an **attribute grammar**.
 - When such a symbol is on top of the stack, execute the semantic action.
- **Problem:** the attributes have to be pushed and popped at a different pace compared to symbols on the parsing stack.
- **Solution:** use an additional stack (the semantic stack) to store the attributes.
 - The semantic actions typically pop semantic records from the semantic stack, do some processing, then push a semantic record on the stack.



Top-down table-driven syntax-directed translation

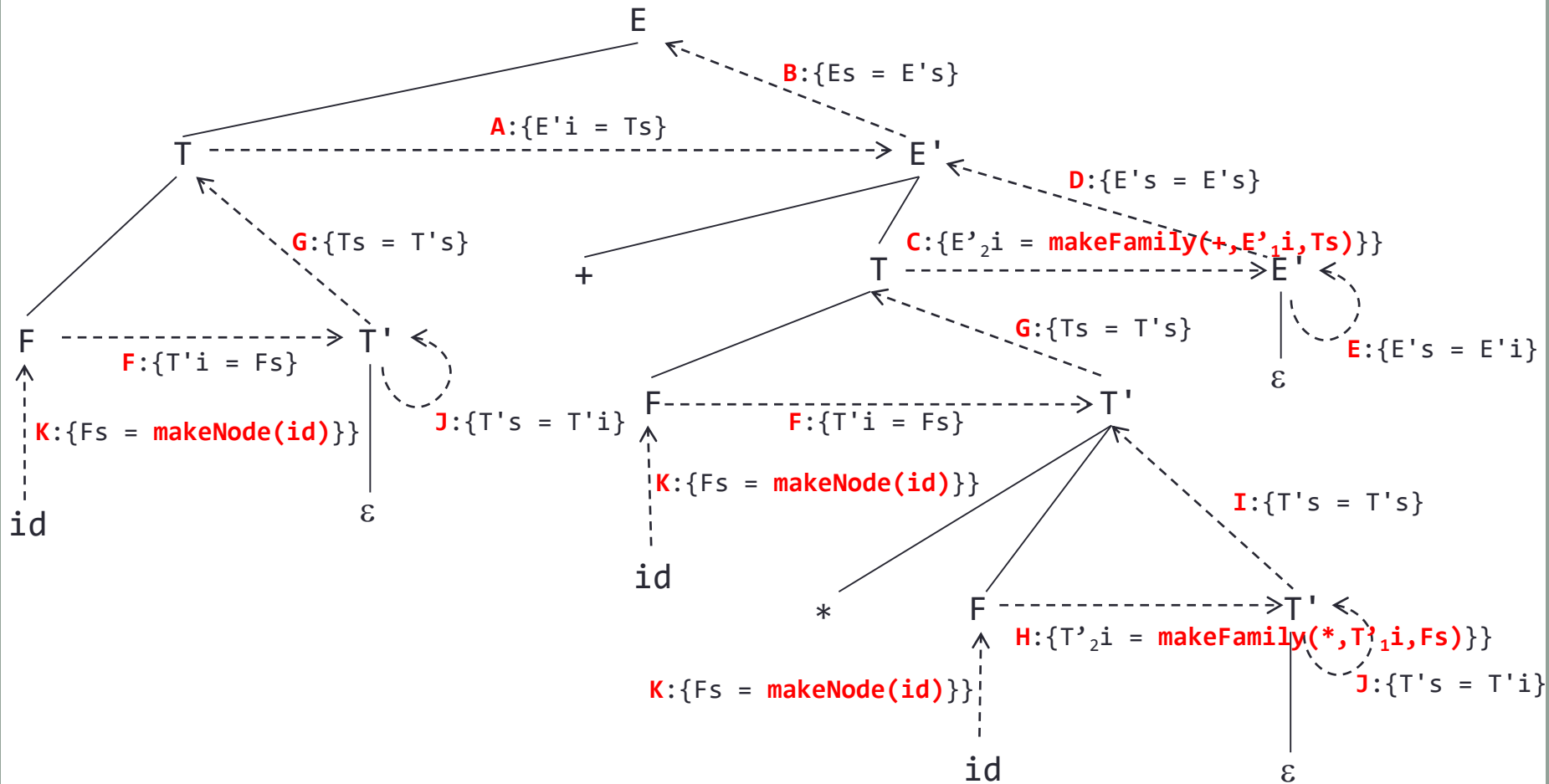
Attribute Grammar	
r1:	$E \rightarrow TAE'B$
r2:	$E'_1 \rightarrow +TCE'_2D$
r3:	$E' \rightarrow \varepsilon E$
r4:	$T \rightarrow FFT'G$
r5:	$T'_1 \rightarrow *FHT'_2I$
r6:	$T' \rightarrow \varepsilon J$
r7:	$F \rightarrow idK$
r8:	$F \rightarrow (E)L$

Semantic Actions	
A:	$\{E'i = Ts\}$
B:	$\{Es = E's\}$
C:	$\{E'_2i = \text{makeFamily}(+, E'_1i, Ts)\}$
D:	$\{E'_1s = E'_2s\}$
E:	$\{E's = E'i\}$
F:	$\{T'i = Fs\}$
G:	$\{Ts = T's\}$
H:	$\{T'_2i = \text{makeFamily}(*, T'_1i, Fs)\}$
I:	$\{T'_1s = T'_2s\}$
J:	$\{T's = T'i\}$
K:	$\{Fs = \text{makeNode}(id)\}$
L:	$\{Fs = Es\}$

	id	()	+	*	\$
E	r1	r1				
E'			r3	r2		r3
T	r4	r4				
T'			r6	r6	r5	r6
F	r7	r8				

Attribute migration

id1+id2*id3\$



Parsing example using semantic stack for attribute migration

	parsing stack	input	action	semantic stack
1	\$E	id ₁ *id ₂ +id ₃ \$	R1	
2	\$BE'AT	id ₁ *id ₂ +id ₃ \$	R4	
3	\$BE'AGT'FF	id ₁ *id ₂ +id ₃ \$	R7	
4	\$BE'AGT'FKid	id ₁ *id ₂ +id ₃ \$		
5	\$BE'AGT'FK	*id ₂ +id ₃ \$	K	F ₁ s[id ₁ .val]
6	\$BE'AGT'F	*id ₂ +id ₃ \$	F	T' ₁ i[id ₁ .val]
7	\$BE'AGT'	*id ₂ +id ₃ \$	R5	T' ₁ i[id ₁ .val]
8	\$BE'AGIT'HF*	*id ₂ +id ₃ \$		T' ₁ i[id ₁ .val]
9	\$BE'AGIT'HF	id ₂ +id ₃ \$	R7	T' ₁ i[id ₁ .val]
10	\$BE'AGIT'HKid	id ₂ +id ₃ \$		T' ₁ i[id ₁ .val]
11	\$BE'AGIT'HK	+id ₃ \$	K	T' ₁ i[id ₁ .val] F ₂ s[id ₂ .val]
12	\$BE'AGIT'H	+id ₃ \$	H	T' ₂ i[id ₁ .val * id ₂ .val]
13	\$BE'AGIT'	+id ₃ \$	R6	T' ₂ i[id ₁ .val * id ₂ .val]
14	\$BE'AGIJ	+id ₃ \$	J	T' ₂ s[id ₁ .val * id ₂ .val]
15	\$BE'AGI	+id ₃ \$	I	T' ₁ s[id ₁ .val * id ₂ .val]
16	\$BE'AG	+id ₃ \$	G	T ₁ s[id ₁ .val * id ₂ .val]
17	\$BE'A	+id ₃ \$	A	E' ₁ i[id ₁ .val * id ₂ .val]

Parsing example using semantic stack for attribute migration

	parsing stack	input	action	semantic stack
18	\$BE'	+id ₃ \$	R2	E' ₁ i[id ₁ .val * id ₂ .val]
19	\$BDE'CT+	+id ₃ \$		E' ₁ i[id ₁ .val * id ₂ .val]
20	\$BDE'CT	id ₃ \$	R4	E' ₁ i[id ₁ .val * id ₂ .val]
21	\$BDE'CGT'FF	id ₃ \$	R7	E' ₁ i[id ₁ .val * id ₂ .val]
22	\$BDE'CGT'FKid	id ₃ \$		E' ₁ i[id ₁ .val * id ₂ .val]
23	\$BDE'CGT'FK	\$	K	E' ₁ i[id ₁ .val * id ₂ .val] F ₃ s[id ₃ .val]
24	\$BDE'CGT'F	\$	F	E' ₁ i[id ₁ .val * id ₂ .val] T' ₃ i[id ₃ .val]
25	\$BDE'CGT'	\$	R6	E' ₁ i[id ₁ .val * id ₂ .val] T' ₃ i[id ₃ .val]
26	\$BDE'CGJ	\$	J	E' ₁ i[id ₁ .val * id ₂ .val] T' ₃ s[id ₃ .val]
27	\$BDE'CG	\$	G	E' ₁ i[id ₁ .val * id ₂ .val] T ₂ s[id ₃ .val]
28	\$BDE'C	\$	C	E' ₂ i[id ₁ .val * id ₂ .val + id ₃ .val]
29	\$BDE'	\$	R3	E' ₂ i[id ₁ .val * id ₂ .val + id ₃ .val]
30	\$BDE	\$	E	E' ₂ s[id ₁ .val * id ₂ .val + id ₃ .val]
31	\$BD	\$	D	E' ₁ s[id ₁ .val * id ₂ .val + id ₃ .val]
32	\$B	\$	B	E ₁ s[id ₁ .val * id ₂ .val + id ₃ .val]
33	\$	\$	accept	E ₁ s[id ₁ .val * id ₂ .val + id ₃ .val]

References

- Fischer, Cytron, Leblanc. *Crafting a Compiler, Chapter 7*. Addison-Wesley. 2010.
- Robert Paul Corbett. [*Static Semantics and Compiler Error Recovery*](#). PhD thesis, University of California Berkeley. 1985.