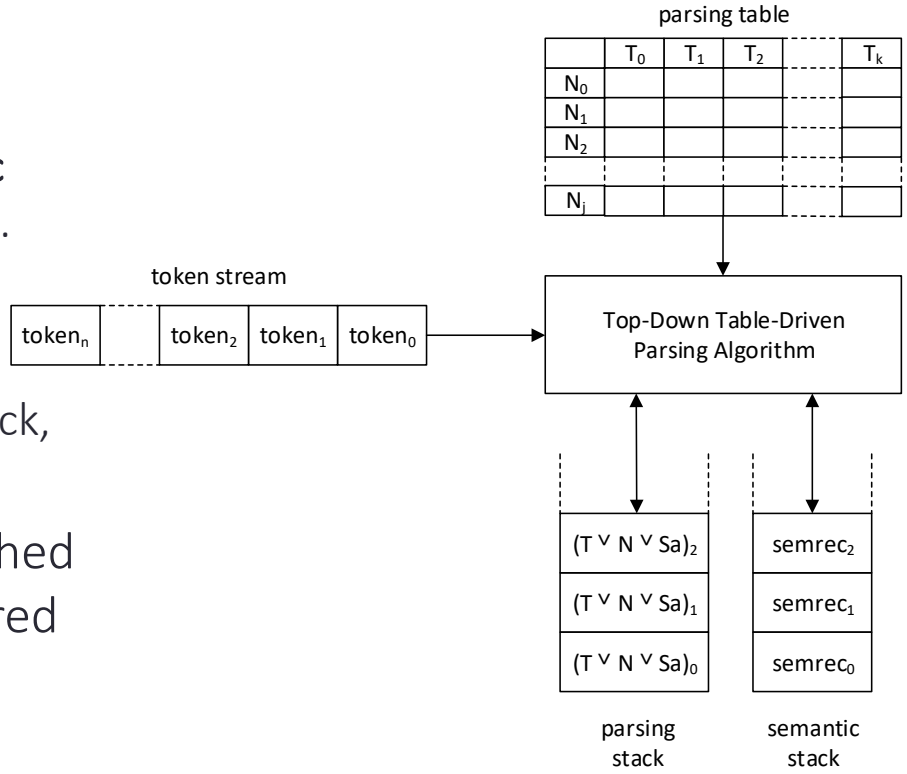
COMPILER DESIGN

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

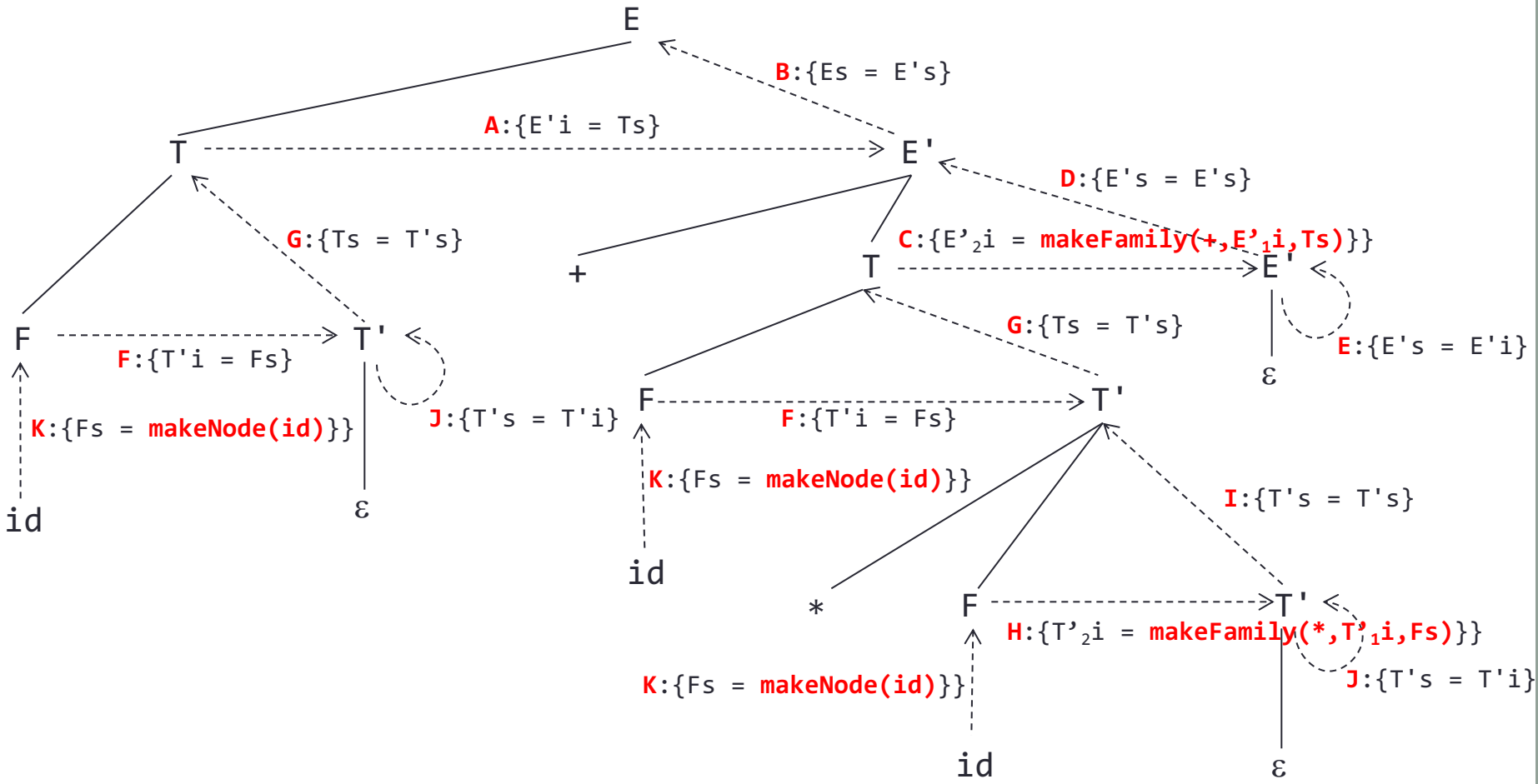
Attribute Grammar	
r1:	$E \rightarrow TAE'B$
r2:	$E'_1 \rightarrow +TCE'_2D$
r3:	$E' \rightarrow \varepsilon E$
r4:	$T \rightarrow FFT'G$
r5:	$T'_1 \rightarrow *FHT'_2I$
r6:	$T' \rightarrow \varepsilon J$
r7:	$F \rightarrow idK$
r8:	$F \rightarrow (E)L$

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

Semantic Actions	
A:	{E'i = Ts}
B:	{Es = E's}
C:	{E'$_2$i = makeFamily(+,E'$_1$i,Ts)}
D:	{E'$_1$s = E'$_2$s}
E:	{E's = E'i}
F:	{T'i = Fs}
G:	{Ts = T's}
H:	{T'$_2$i = makeFamily(*,T'$_1$i,Fs)}
I:	{T'$_1$s = T'$_2$s}
J:	{T's = T'i}
K:	{Fs = makeNode(id)}
L:	{Fs = Es}

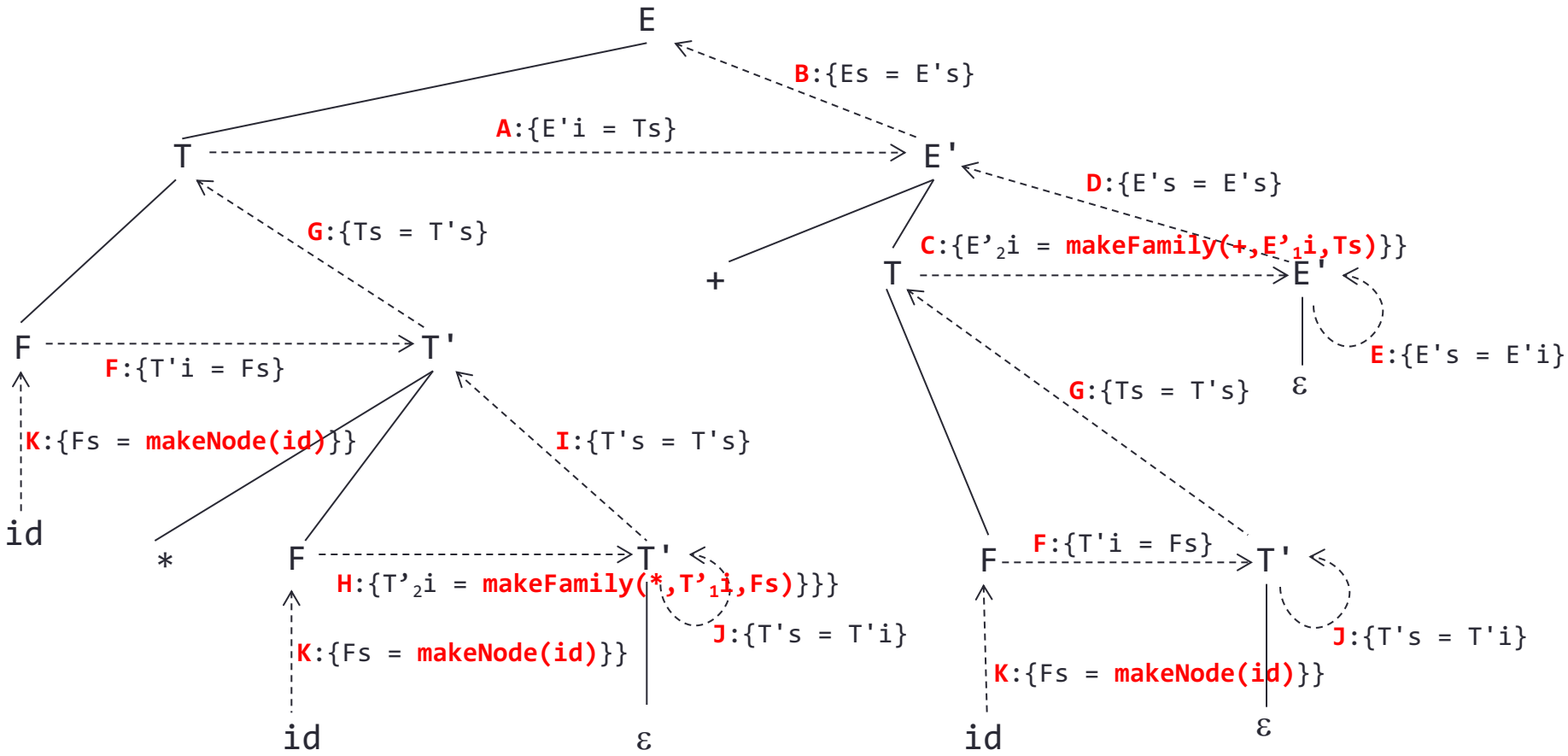
Attribute migration

id1+id2*id3$



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	$F_1s[id_1.val]$
6	$BE'AGT'F	$*id_2+id_3$	F	$T'_1i[id_1.val]$
7	$BE'AGT'	$*id_2+id_3$	R5	$T'_1i[id_1.val]$
8	$BE'AGIT'HF*	$*id_2+id_3$		$T'_1i[id_1.val]$
9	$BE'AGIT'HF	id_2+id_3	R7	$T'_1i[id_1.val]$
10	$BE'AGIT'HKid	id_2+id_3		$T'_1i[id_1.val]$
11	$BE'AGIT'HK	$+id_3$	K	$T'_1i[id_1.val]$ $F_2s[id_2.val]$
12	$BE'AGIT'H	$+id_3$	H	$T'_2i[id_1.val * id_2.val]$
13	$BE'AGIT'	$+id_3$	R6	$T'_2i[id_1.val * id_2.val]$
14	$BE'AGIJ	$+id_3$	J	$T'_2s[id_1.val * id_2.val]$
15	$BE'AGI	$+id_3$	I	$T'_1s[id_1.val * id_2.val]$
16	$BE'AG	$+id_3$	G	$T_1s[id_1.val * id_2.val]$
17	$BE'A	$+id_3$	A	$E'_1i[id_1.val * id_2.val]$

Parsing example using semantic stack for attribute migration

	parsing stack	input	action	semantic stack
18	$BE'	$+id_3$	R2	$E'_1i[id_1.val * id_2.val]$
19	$BDE'CT+	$+id_3$		$E'_1i[id_1.val * id_2.val]$
20	$BDE'CT	id_3	R4	$E'_1i[id_1.val * id_2.val]$
21	$BDE'CGT'FF	id_3	R7	$E'_1i[id_1.val * id_2.val]$
22	$BDE'CGT'FKid	id_3		$E'_1i[id_1.val * id_2.val]$
23	$BDE'CGT'FK	$$	K	$E'_1i[id_1.val * id_2.val]\ F_3s[id_3.val]$
24	$BDE'CGT'F	$$	F	$E'_1i[id_1.val * id_2.val]\ T'_3i[id_3.val]$
25	$BDE'CGT'	$$	R6	$E'_1i[id_1.val * id_2.val]\ T'_3i[id_3.val]$
26	$BDE'CGJ	$$	J	$E'_1i[id_1.val * id_2.val]\ T'_3s[id_3.val]$
27	$BDE'CG	$$	G	$E'_1i[id_1.val * id_2.val]\ T_2s[id_3.val]$
28	$BDE'C	$$	C	$E'_2i[id_1.val * id_2.val + id_3.val]$
29	$BDE'	$$	R3	$E'_2i[id_1.val * id_2.val + id_3.val]$
30	$BDE	$$	E	$E'_2s[id_1.val * id_2.val + id_3.val]$
31	$BD	$$	D	$E'_1s[id_1.val * id_2.val + id_3.val]$
32	$B	$$	B	$E_1s[id_1.val * id_2.val + id_3.val]$
33	$	$$	accept	$E_1s[id_1.val * id_2.val + id_3.val]$

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.