CSE443 Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Team status updates

Items

- "How does a shift-reduce parser know when to shift and when to reduce?" [p 242]
- "...by maintaining states to keep track of where we are in a parse."
- Each state is a set of items.
- An item is a grammar rule annotated with a dot, •, somewhere on the RHS.

Rules and items

A -> X Y Z
A -> • X Y Z
A -> X • Y Z
A -> X Y • Z
A -> X Y Z •

A -> ε
A -> •

The • shows where in a rule we might be during a parse.

Building the finite control for a bottom-up parser

- Build a finite state machine, whose states are sets of items
- Build a table (M) incorporating shift/reduce decisions

Augment grammar

Given a grammar

$G = (N,T,P,S)$

we augment to a grammar

$G' = (N \cup \{S'\},T,P \cup \{S' \to S\},S')$, where $S' \notin N$

G' has exactly one rule with S' on left.

We need two operations to build our finite state machine

CLOSURE(I)

GOTO(I,X)

CLOSURE(I)

- I is a set of items
- CLOSURE(I) fixed point construction

$CLOSURE_0(I) = I$

repeat {

$CLOSURE_{i+1}(I) =$

$CLOSURE_i(I) \cup \{ B \text{->} \bullet\gamma \mid A \text{->} \alpha\bullet B\beta \in CLOSURE_i(I) \text{ and } B \text{->} \gamma \in P \}$

} until $CLOSURE_{i+1}(I) = CLOSURE_i(I)$

CLOSURE(I)

- I is a set of items
- CLOSURE(I) fixed point construction

Intuition: an item like A -> X • Y Z conveys that we've already seen X, and we're expecting to see a Y followed by a Z.

The closure of this item is all the other items that are relevant at this point in the parse.

For example, if Y -> R S T is a production, then Y -> • R S T is in the closure because if the next thing in the input can derive from Y, it can derive from R.

GOTO(I,X)

- GOTO(I,X) is the closure of the set of items $A \rightarrow \alpha X \bullet \beta$ s.t. $A \rightarrow \alpha \bullet X \beta \in I$
- GOTO(I,X) construction for G' (figure 4.32):

```
set-of-items CLOSURE(I) {
    J = I
    repeat {
        for each item A -> α•Bβ ∈ J
            for each production B -> γ ∈ P
                if B->•γ not already in J
                    add B->•γ to J
    } until no more items are added to J
    return J
}
```

Building the LR(0) automaton

```
void items(G') {
   C = { CLOSURE( { S' -> •S } ) }
   repeat {
      for each set of items I ∈ C and
      for each grammar symbol X ∈ (N∪T)
      if ( GOTO(I,X) is not empty and not already in C )
         add GOTO(I,X) to C
   } until no new sets of items are added to C
}
```

C is a set of sets of items

Example [p 245]

Grammar G	Augmented Grammar G'
	S' -> E
E -> E + T	E -> E + T
E -> T	E -> T
T -> T * F	T -> T * F
T -> F	T -> F
F -> (E)	F -> (E)
F -> id	F -> id

Compute items(G')

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

SET OF ITEMS (I)	i	$CLOSURE_i(I)$
{ S' -> • E }	0	{ S' -> • E }

Compute items(G')

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

SET OF ITEMS (I)	i	$CLOSURE_i(I)$
{ S' -> • E }	0	{ S' -> • E }
	1	$CLOSURE_0(I)$ ∪ { E -> • E + T , E -> • T }

Compute items(G')

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

SET OF ITEMS (I)	i	$CLOSURE_i(I)$
{ S' -> • E }	0	{ S' -> • E }
	1	$CLOSURE_0(I) \cup$ { E -> • E + T , E -> • T }
	2	$CLOSURE_1(I) \cup$ { T -> • T * F , T -> • F }

Compute items(G')

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

SET OF ITEMS (I)	i	$\text{CLOSURE}_i(I)$
{ S' -> • E }	0	{ S' -> • E }
	1	$\text{CLOSURE}_0(I)$ ∪ { E -> • E + T , E -> • T }
	2	$\text{CLOSURE}_1(I)$ ∪ { T -> • T * F , T -> • F }
	3	$\text{CLOSURE}_2(I)$ ∪ { F -> • (E) , F -> • id }

Compute items(G')

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

SET OF ITEMS (I)	i	$CLOSURE_i(I)$
{ S' -> • E }	0	{ S' -> • E }
	1	$CLOSURE_0(I)$ ∪ { E -> • E + T , E -> • T }
	2	$CLOSURE_1(I)$ ∪ { T -> • T * F , T -> • F }
	3	$CLOSURE_2(I)$ ∪ { F -> • (E) , F -> • id }
	4	$CLOSURE_3(I)$ ∪ ∅

Terminology

- Kernel items: S' -> • S and all items with • not at left edge
- Non-kernel items: all items with • at left edge, except S' -> • S

This gives us the first state of the finite state machine, I_0

I_0

S' -> • E
E -> • E + T
E -> • T
T -> • T * F
T -> • F
F -> • (E)
F -> • id

kernel item

non-kernel items are computed from CLOSURE(kernel), and therefore do not need to be explicitly stored

Next we compute $GOTO(I_0,X)\ \forall\ X \in N \cup T$

$N \cup T = \{ E , T , F , + , * , (,) , id \}$

N.B. - augmented start symbol S' can be ignored

$GOTO(I_0,E) = CLOSURE(\{ S' \rightarrow E \bullet , E \rightarrow E \bullet + T \})$

$= \{ S' \rightarrow E \bullet , E \rightarrow E \bullet + T \}$

N.B. there is no non-terminal after the •, so no new items are added by CLOSURE operation

I_1 $\boxed{\begin{array}{l} S' \rightarrow E \bullet \\ E \rightarrow E \bullet + T \end{array}}$

only kernel items

$GOTO(I_0, T) = CLOSURE(\{ E \to T \bullet, T \to T \bullet * F \})$

$= \{ E \to T \bullet, T \to T \bullet * F \}$

N.B. there is no non-terminal after the •, so no new items are added by CLOSURE operation

I_2 $\boxed{\begin{array}{l} E \to T \bullet \\ T \to T \bullet * F \end{array}}$

only kernel items

$GOTO(I_0,F) = CLOSURE(\{ T \to F \bullet \})$

$= \{ T \to F \bullet \}$

N.B. there is no non-terminal after the •, so no new items are added by CLOSURE operation

I_3 $\boxed{T \to F \bullet}$

only kernel items

N.B. there is a non-terminal after the •, so new items are added by CLOSURE operation

GOTO(I_0, '(') = CLOSURE({ F -> (• E) })

= { F -> (• E) } ∪ { E -> • E + T , E -> • T } ∪ { T -> • T * F , T -> • F } ∪ { F -> • (E) , F -> • id }

I_4	F -> (• E)	kernel item
	E -> • E + T E -> • T T -> • T * F T -> • F F -> • (E) F -> • id	non-kernel items

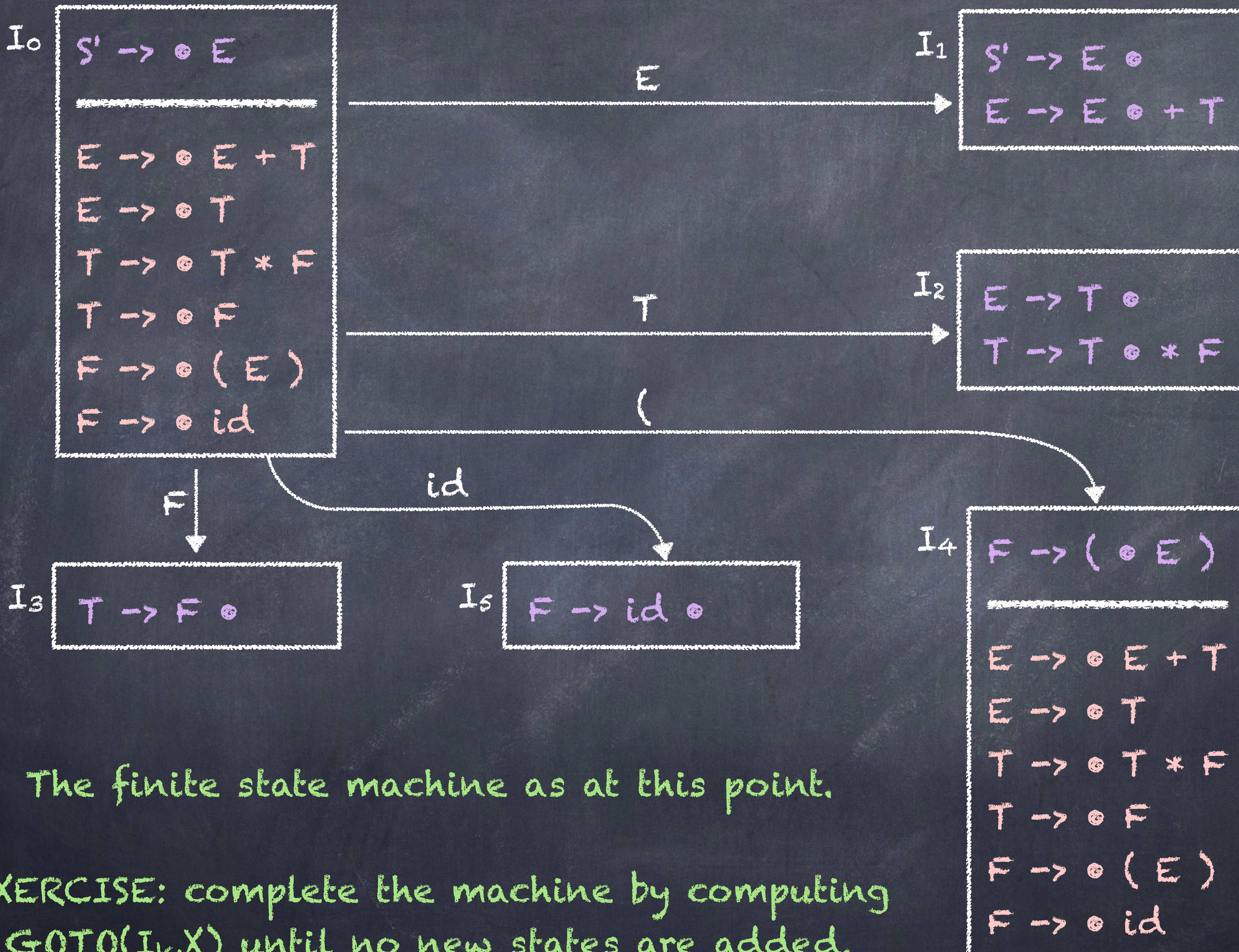
$GOTO(I_0, id) = CLOSURE(\{ F \to id \bullet \})$

N.B. there is no non-terminal after the •, so no new items are added by CLOSURE operation

$= \{ F \to id \bullet \}$

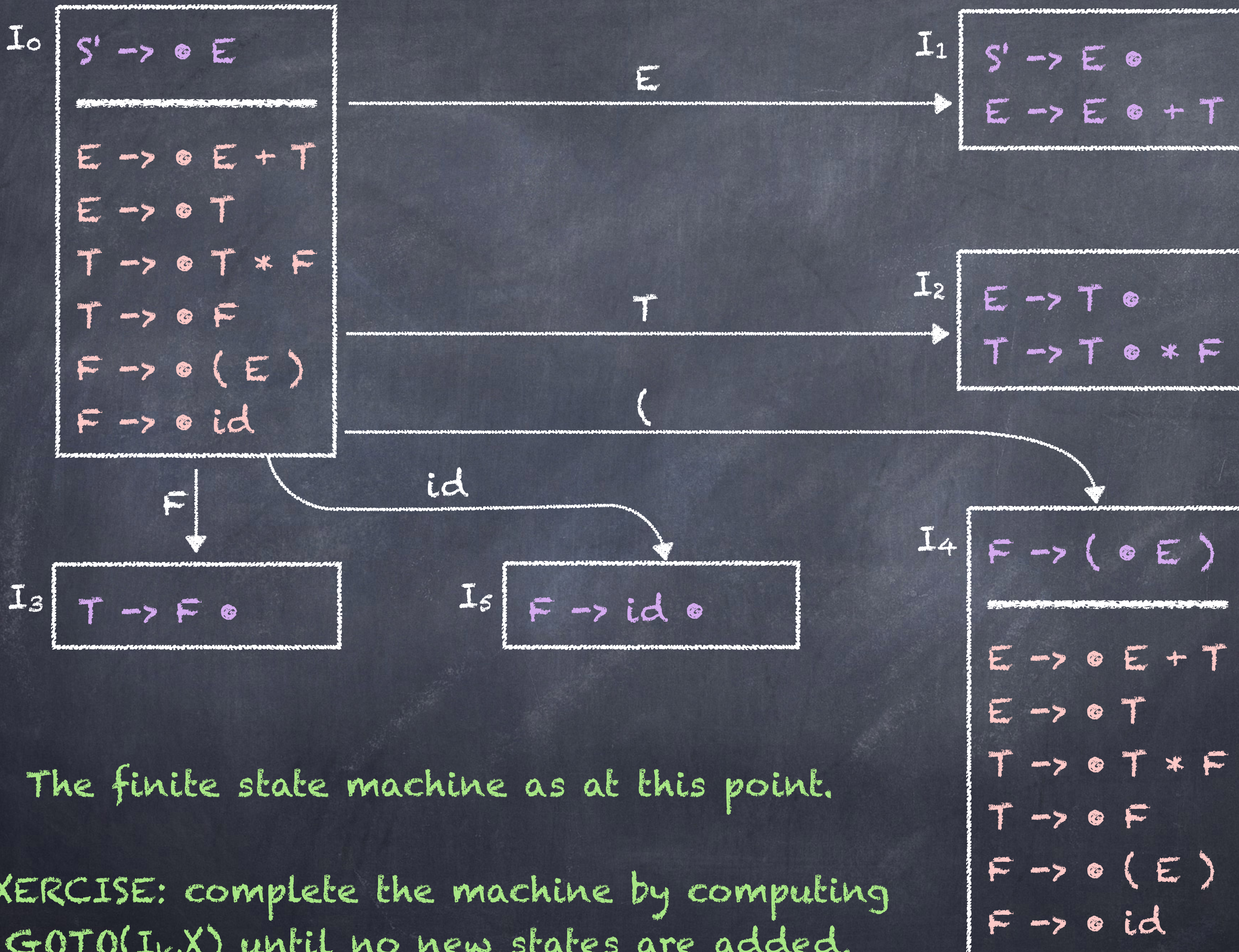
I_5 $\boxed{F \to id \bullet}$ only kernel items

$GOTO(I_0 , ')') = GOTO(I_0 , +) = GOTO(I_0 , *) = GOTO(I_0 , \$) = \varnothing$



The finite state machine as at this point.

EXERCISE: complete the machine by computing $GOTO(I_k,X)$ until no new states are added.



The finite state machine as at this point.

EXERCISE: complete the machine by computing $GOTO(I_k, X)$ until no new states are added.

I_0

S' -> • E

E -> • E + T
E -> • T
T -> • T * F
T -> • F
F -> • (E)
F -> • id

I_1

S' -> E •
E -> E • + T

I_2

E -> T •
T -> T • * F

I_3

T -> F •

I_5

F -> id •

I_4

F -> (• E)

E -> • E + T
E -> • T
T -> • T * F
T -> • F
F -> • (E)
F -> • id

$I_0 \xrightarrow{E} I_1$, $I_0 \xrightarrow{T} I_2$, $I_0 \xrightarrow{F} I_3$, $I_0 \xrightarrow{id} I_5$, $I_0 \xrightarrow{(} I_4$

S' -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> (E)
F -> id

Compute GOTO(I_1,X) for each X in { +, *, '(', ')', id, E, T, F, $ }