Figura 3.5: L'unico albero di derivazione per la stringa con la nuova grammatica

in questo caso il comando fra il **then** e l'**else** è un comando "matched" e il comando dopo l'**else** è "unmatched".

In questo modo ogni comando che appare tra un **then** ed un **else** non può finire con un **then** non associato seguito da un altro comando qualsiasi. Ecco la nuova grammatica:

$$
\begin{array}{rcl}
< C > & \longrightarrow & < MC > \mid < UC > \\
< MC > & \longrightarrow & \textbf{if} < E > \textbf{then} < MC > \textbf{else} < MC > \\
 & \mid & < \text{ALTRI} > \\
< UC > & \longrightarrow & \textbf{if} < E > \textbf{then} < C > \\
 & \mid & \textbf{if} < E > \textbf{then} < MC > \textbf{else} < UC >
\end{array}
$$

In Figura 3.5 è disegnato l'unico albero di derivazione per la stringa incriminata. Questa volta non si può costruire un albero simile a quello di Figura 3.4 perché fra il primo **then** e l'**else** non possiamo mettere un comando "unmatched" come un condizionale senza l'**else**.

3.3 Esprimere l'associatività e la precedenza degli operatori

3.4 Top-Down Parsing

3.5 Bottom-Up Parsing

In questa sezione introduciamo una tecnica di analisi sintattica bottom-up che prende il nome di *shift-reduce parsing* o analisi appila-riduci. Un metodo generale per questo tipo di parsing è il cosiddetto LR parsing che è usato in molti generatori automatici di analizzatori sintattici.

L'analisi appila-riduci si occupa di costruire il parse tree della stringa di token che gli viene data in input. La costruzione parte dalle foglie dell'albero e continua con l'accorpamento di sottoalberi fino ad arrivare alla radice del parse tree cercato.

Questo processo può equivalentemente essere pensato come la riduzione di una stringa w di simboli terminali in input al simbolo iniziale della grammatica. Ad ogni passo di questa riduzione una sottostringa, della forma

sentenziale corrente, che fa match con la parte destra di una produzione della grammatica viene riscritta nel simbolo non terminale a sinistra della produzione in questione. Se ad ogni passo di riduzione la sottostringa viene scelta correttamente allora la sequenza di riduzioni fatte termina con una forma sentenziale che contiene solo il simbolo iniziale della grammatica e tutta la sequenza di riduzione corrisponde ad una derivazione rightmost della stringa w a partire dal simbolo iniziale.

Esempio 3.8 *Chiariamo meglio questo processo con l'aiuto di un esempio. Consideriamo la seguente grammatica:*

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$

e consideriamo la stringa di input $w = abbcde$. Questa stringa può essere ridotta a S tramite i seguenti passi di riduzione:

$$\begin{array}{lll} a\underline{b}bcde & A \rightarrow b \\ a\underline{Abc}de & A \rightarrow Abc \\ aA\underline{d}e & B \rightarrow d \\ \underline{aABe} & S \rightarrow aABe \\ S \end{array}$$

dove la parte sottolineata è la sottostringa uguale alla parte destra della produzione, indicata a fianco, che utilizziamo nella riduzione.

Il procedimento generale è quello di cercare, ad ogni passo, una sottostringa uguale alla parte destra di una produzione della grammatica. C'è la possibilità che ne esista più di una: ad esempio al primo passo entrambe le b in seconda e terza posizione e la d in quinta posizione possono essere ridotte. Nell'esempio abbiamo scelto la b in seconda posizione e l'abbiamo ridotta ad A. Il punto chiave è fare la scelta giusta.

Al secondo passo dobbiamo effettuare un'altra scelta poichè b, Abc e d sono tutte passibili di riduzione. Andando avanti, e facendo le scelte giuste, siamo riusciti a ricostruire la seguente derivazione rightmost per la stringa data:

$$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$$

3.5.1 Handle

Informalmente una *handle* (maniglia) di una stringa è una sottostringa che è esattamente uguale alla parte destra di una certa produzione e la cui

riduzione al simbolo a sinistra della produzione rappresenta un passo lungo una derivazione rightmost rovesciata.

È importante far notare subito che in molti casi la sottostringa più a sinistra che è uguale alla parte destra di una certa produzione non è una handle perché una riduzione tramite questa produzione produce una stringa dalla quale non esiste una sequenza di riduzioni che portano al simbolo iniziale della grammatica. Per questo motivo è importante dare una definizione precisa di "handle".

Sia γ una forma sentenziale destra, cioè esiste una derivazione del tipo $S \stackrel{*}{\Rightarrow}_{rm} \gamma$. Diciamo che una handle di γ è formata da una produzione $A \to \beta$ e da una posizione in γ dove appare la stringa β tali che se quest'ultima viene rimpiazzata con A allora il risultato è la forma sentenziale destra precedente a γ in una derivazione rightmost di γ stessa.

In altre parole, se $S \stackrel{*}{\Rightarrow}_{rm} \alpha A w \Rightarrow_{rm} \alpha\beta w$ allora la produzione $A \to \beta$ e la posizione successiva all'occorrenza di α è una handle per la forma sentenziale destra $\alpha\beta w$. La stringa w a destra della handle contiene solo simboli terminali.

Finora abbiamo detto "una handle" invece che "la handle" perché se la grammatica che consideriamo è ambigua allora possono esistere più derivazioni rightmost per la stessa stringa e, quindi, diverse handle all'interno della stessa forma sentenziale destra. Tuttavia se la grammatica non è ambigua sappiamo che per ogni forma sentenziale destra esiste una sola derivazione rightmost che la genera a partire dal simbolo iniziale e, quindi, ad ogni passo di riduzione la handle è unica.

Nell'esempio precedente *abbcde* è una forma sentenziale destra la cui handle è la produzione $A \to b$ e la posizione 2. Allo stesso modo, *aAbcde* ha come handle la produzione $A \to Abc$ e la posizione 2. Alcune volte possiamo dire direttamente "la sottostringa β è una handle di $\alpha\beta w$" se la produzione $A \to \beta$ e la posizione di β sono chiare dal contesto.

Figura 3.6 visualizza la handle β in un albero di derivazione (incompleto) di $\alpha\beta w$. La handle è il sottoalbero più a sinistra che è completo e che consiste di un nodo interno e di tutti i suoi figli. Graficamente, A è il nodo interno più in basso e a sinistra che ha tutti i figli nell'albero. Ridurre β ad A in $\alpha\beta w$ può essere pensato come ad una "potatura" dell'albero, cioè alla rimozione di tutti i figli di A.

Esempio 3.9 *Consideriamo la seguente grammatica:*

$$
\begin{array}{lllll}
(1) & E & \to & E + E \\
(2) & E & \to & E * E \\
(3) & E & \to & (E) \\
(4) & E & \to & \mathbf{id}
\end{array}
$$

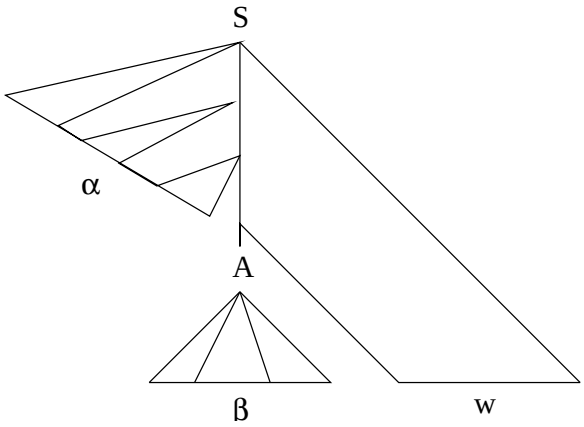

Figura 3.6: La handle $A \to \beta$ nel parse tree per $\alpha\beta w$.

e la seguente derivazione rightmost:

$$\begin{array}{lll} E & \Rightarrow_{rm} & \underline{E + E} \\ & \Rightarrow_{rm} & E + \underline{E * E} \\ & \Rightarrow_{rm} & E + E * \underline{\mathbf{id}_3} \\ & \Rightarrow_{rm} & E + \underline{\mathbf{id}_2} * \mathbf{id}_3 \\ & \Rightarrow_{rm} & \underline{\mathbf{id}_1} + \mathbf{id}_2 * \mathbf{id}_3 \end{array}$$

Abbiamo numerato gli **id** *per convenienza di notazione e la parte di stringa sottolineata in ogni forma sentenziale destra è una handle. Ad esempio* $\mathbf{id}_1$ *è una handle in* $\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3$ *poiché* **id** *è la parte destra della produzione* $E \to \mathbf{id}$ *e rimpiazzando* $\mathbf{id}_1$ *con* E *otteniamo la forma sentenziale precedente* $E + \mathbf{id}_2 * \mathbf{id}_3$*. Notiamo, come abbiamo già detto, che in ogni caso la stringa che appare a destra della handle è sempre formata solo da simboli terminali.*

Sappiamo già che questa grammatica è ambigua. La seguente è un'altra derivazione rightmost, con relativa indicazione delle handle, per la stringa considerata prima:

$$\begin{array}{lll} E & \Rightarrow_{rm} & \underline{E * E} \\ & \Rightarrow_{rm} & E * \underline{\mathbf{id}_3} \\ & \Rightarrow_{rm} & \underline{E + E} * \mathbf{id}_3 \\ & \Rightarrow_{rm} & E + \underline{\mathbf{id}_2} * \mathbf{id}_3 \\ & \Rightarrow_{rm} & \underline{\mathbf{id}_1} + \mathbf{id}_2 * \mathbf{id}_3 \end{array}$$

Consideriamo la forma sentenziale destra $E + E + \mathbf{id}_3$*. In questa ultima derivazione* $E + E$ *è una handle, ma anche* $\mathbf{id}_3$ *è una handle della stessa forma sentenziale considerando l'altra derivazione rightmost.*

3.5.2 Handle pruning

Possiamo pensare al processo di costruire una derivazione rightmost rovesciata in termini di handle pruning (potatura delle maniglie) dal parse tree della forma sentenziale destra che dobbiamo ridurre. Partiamo dalla stringa di terminali (token) w che vogliamo analizzare. Se w è una stringa generata dalla grammatica a cui facciamo riferimento allora $w = \gamma_n$ dove γ_n l'n-esima forma sentenziale destra di una qualche derivazione rightmost ancora sconosciuta:

$$S = \gamma_0 \Rightarrow_{rm} \gamma_1 \Rightarrow_{rm} \cdots \Rightarrow_{rm} \gamma_{n-1} \Rightarrow_{rm} \gamma_n = w$$

Per ricostruire questa derivazione in ordine inverso dobbiamo trovare la handle β_n in γ_n e rimpiazzare β_n con la parte sinistra di una produzione $A_n \rightarrow \beta_n$. Otterremo in questo modo γ_{n-1}. *Anocra non sappiamo come fare a trovare le maniglie, ma presto vedremo dei modi per farlo.*

Questo processo viene ripetuto fino a quando non risaliamo a $\gamma_0 = S$. A questo punto l'analisi termina con successo.

3.5.3 Shift-Reduce Parsing

Ci sono due problemi principali che vanno risolti durante il parsing tramite potatura delle maniglie:

1. Trovare la sottostringa β che deve essere ridotta
2. Scegliere quale produzione utilizzare nel caso ci siano più produzioni con la stessa parte destra β, ma con simbolo a sinistra diverso.

Prima di occuparci in dettaglio di questi problemi vediamo che tipo di strutture dati conviene utilizzare per il parsing *shift-reduce* (analisi impila-riduci). È utile utilizzare uno stack che possa contenere simboli della grammatica (sia terminali sia non) e un buffer di input che possa contenere la stringa w che deve essere utilizzata. Come nel caso del top-down parsing usiamo un simbolo speciale \$ per marcare il fondo dello stack e la fine della stringa di input.

Inizialmente lo stack deve essere vuoto e tutta la stringa deve essere nell'input:

STACK	INPUT
\$	w \$

L'analizzatore shift-reduce opera nel seguente modo: mette sullo stack zero o più simboli di input, prelevandoli dal buffer, fino a quando non riconosce che sulla testa dello stack c'è una handle β. A questo punto riduce

la stringa β eliminandola dallo stack e mettendo al suo posto (in testa) il simbolo non terminale a sinistra della produzione della handle.

Questo processo viene iterato fino a che non viene incontrato un errore (sintattico) oppure si raggiunge la configurazione di accettazione:

STACK	INPUT
\$S	\$

Esempio 3.10 *Vediamo tutti i passi che dovrebbe fare un parser shift-reduce che analizza* $\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3$*, fa riferimento alla grammatica dell'esempio 3.9 e ricostruisce la prima derivazione rightmost presentata nell'esempio. Ovviamente esiste un'altra sequenza di passi che un parser shift-reduce può fare per analizzare questa stringa con la grammatica data, cioè quella corrispondente all'altra derivazione rightmost.*

	STACK	*INPUT*	*AZIONE*
(1)	$\$$	$\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3\ \$$	*shift*
(2)	$\$\ \mathbf{id}_1$	$+\mathbf{id}_2 * \mathbf{id}_3\ \$$	*riduci con* $E \to \mathbf{id}$
(3)	$\$\ E$	$+\mathbf{id}_2 * \mathbf{id}_3\ \$$	*shift*
(4)	$\$\ E+$	$\mathbf{id}_2 * \mathbf{id}_3\ \$$	*shift*
(5)	$\$\ E + \mathbf{id}_2$	$*\mathbf{id}_3\ \$$	*riduci con* $E \to \mathbf{id}$
(6)	$\$\ E + E$	$*\mathbf{id}_3\ \$$	*shift*
(7)	$\$\ E + E*$	$\mathbf{id}_3\ \$$	*shift*
(8)	$\$\ E + E * \mathbf{id}_3$	$\$$	*riduci con* $E \to \mathbf{id}$
(9)	$\$\ E + E * E$	$\$$	*riduci con* $E \to E * E$
(10)	$\$\ E + E$	$\$$	*riduci con* $E \to E + E$
(11)	$\$\ E$	$\$$	*accetta*

Le operazioni principali sono shift e reduce, ma ce ne sono altre due possibili:

1. *accetta* in cui il parser si ferma e annuncia che il parsing ha avuto successo

2. *errore* in cui il parser ha trovato un errore sintattico. Si ferma e chiama una routine di gestione degli errori.