Nel 7 step il linguaggio viene esteso con nuovi costrutti, in particolare mi ha dato la possibilità di definire puntatori, di dereferenziarli e così via sostanzialmente ci dice che a livello di sintassi tutti i puntatori devono essere definiti prima dell inizio del programma, quindi come con le procedure nello step precedente, il livello del puntatore, quindi la label segue le regole delle variabili normali, quindi si inizia con la h è di tipo high altrimenti è low, però il livello del contenuto puntato dal puntatore è dichiarato nella dichiarazione quindi, se scrivo:

declare ref hp : low;

sto dichiarando un puntatore high che punta a un contenuto a qualcosa di tipo low, altrimenti se scrivo:

declare red lp : high;

sto dichiarando un puntatore low che punta un tipo high.

Si può creare un puntatore alto a qualcosa di basso, lo si fa a puntare ad h1 e poi dereferenziare hp facendolo puntare ad l1.

```
declare ref hp : low;
ref hp = h1;
l1 = deref(hp);
```

Nello step 8 introduciamo gli array e ci sono due nuove regole per il typesystem degli array sia per quanto riguarda l' assegnamento ad una variabile di una porzione dell'array sia per quanto riguarda l' assegnamento di un valore di una qualche espressione a una porzione dell'array. La label del contenuto dell'array è contenuto nella dichiarazione come prima. Allora qui non abbiamo interi, abbiamo solo i booleani e quindi possono avere solo due posizioni nell'array che sono true e false essendo gli indici dell'array booleani. Come prima possono darli solo all'inizio del programma e poi utilizzarli nel seguito. Ho dichiarato un array basso con contenuti bassi e ho sfruttato il fatto che abbiamo solo valori booleani per riuscire a indicizzare l'array quindi i valori di h1 e h2 etc... e quindi il mio ragionamento è stato quindi, se h1 per esempio è true, allora il valore true dell'array sarà true quello opposto invece sarà false e poi quindi creo una variabile prova che gli passa il valore true quindi che se h1 era true allora l1 sara true se invece h1 era false allora prova sarà false e di conseguenza l1 sarà false. Così riesco ad avere sempre che l1 è sempre uguale ad h1.

```
declare array larray : low;
larray[h1] = true;
larray[!h1] = false;
prova = larray[true];
if(prova) l1 = true; else l1 = false;
```

L'idea è stata quella di sfruttare il fatto che l'array indicizzato solo da due booleani da cui si deduce il valore di h1 però, come vede, non accade mai, per esempio, che assegno un valore high tipo h1 a l1 o a porzioni dell'array.

Nello step 9 non si estende il linguaggio con nessun costrutto aggiuntivo anzi abbiamo il linguaggio iniziale. Lo step viene eseguito una sola volta e sempre con gli stessi valori iniziali di h1 h2 fino ad h6. Mentre negli step precedenti, innanzitutto veniva eseguito due volte, ma con valori sempre casuali. Abbiamo deciso di sfruttare dei cicli while infiniti nei determinati casi in modo tale ad ogni iterazione riuscire a scoprire un valore nuovo fino ad ottenerli tutti e sei.

while(h1) h1 = h1 h1 = h1 è un assegnamento per fare uno step senza però modificare nulla, così mandando in esecuzione abbiamo visto se h1 fosse true o false perché in caso true entrava in loop per poi veniva dato un messaggio di errore dicendo che era passato troppo tempo. L'idea era proprio quella di contare gli step che impiega il programma. In questo caso restituisce Maximum execution time exceeded cioè è un while infinito h1 è sempre true e siccome il valore di h1 non cambia mai sappiamo che anche il prossimo sarà true quindi posso fare l1 = true per rifare la stessa cosa su h2. l1 = true; while(h2) h2 = h2; In questo caso non è successo, non era un come prima, non è successo la stessa cosa di prima che il ciclo ad essere infinito ma termina non solo step perché segue solo un istruzione per cui l2 sarà false e così via. In questo step non c'era da sfruttare né un particolare costrutto del linguaggio né delle regole del typesystem bensì delle informazioni aggiuntive come questa, il fatto che le variabili sono inizializzte sempre allo stesso modo, che i run avvengono in una volta sola

abbiamo sempre la stessa informazione e questa volta il loop con le guardie alte non sono permessi, non sono permessi nemmeno i loop all'interno dei branch di un if con le guardie alte, quindi il gioco di prima non si può fare. Abbiamo sfruttato gli if in particolare abbiamo contato il numero di step che venivano eseguiti e quindi praticamente se h1 è true allora viene eseguito uno step, altrimenti se false vengono eseguiti due step e di conseguenza riusciamo a capire effettivamente qual è il valore di h1 e lo assegniamo ad l1 e possiamo eseguirlo anche per h2 h3 e così via.

Il ragionamento è esattamente come prima il fatto di contare il numero di step e sfruttare il fatto che il valore delle variabili è sempre lo stesso che viene eseguito una sola volta e così via. L'unica cosa è che non possiamo più usare le altre, ma possiamo usare gli if con le guardie alte, anche se all'interno degli if non posso mettere a non posso mettere un loop ma questo non è importante perché possiamo fare esattamente lo stesso a ragionamento di prima, quindi se io ora eseguo vedo che l' esecuzione è terminata in un solo step, quindi sarò nel ramo then dell' if e quindi h1 sarà true, per cui l1 è true e posso ripetere lo stesso ragionamento su h2.

```
l1=true;
l2=false;
if(h2) h2=h2;
else{h2=h2;h2=h2}
```