Voici la structure choisie pour représenter un noeud de l'arbre:

```
typedef struct node
{
    char *ruleName;
    struct node *child;
    struct node *brother;
    char * start;
    int len;
    char *value;
} Node;
```

Comme les noeuds peuvent avoir un nombre quelconque de fils, nous avons choisi d'implementer une liste chainée de frères (accessible par `brother`) pour chaque noeud. Ainsi, pour un Node `n`, son premier fils se situe à `n.child`, et les fils suivants sont accessibles par la liste chainée `n.child.brother`.

Processus de création/validation de l'arbre:
Pour pouvoir valider un noeud intermediaire:

- Il créé ses fils
- Il valide ses fils
    - Si la validation réussie, tout va bien
    - Sinon, on supprime les fils de l'arbre

La validation est donc récursive, car la validation d'un noeud intermediaire entraine la validation de ses fils, qui eux meme valideront les leurs.

Cas particuliers a prendre en compte:

- un fils non validé n'amene pas forcement a la suppression de tout ses freres. Exemple: la validation de `* segment` ne dépend pas de la reussite de la validation de `segment`, car `*` implique 0 ou n répétitions. Dans notre architecture, il faudra juste supprimer le/les derniers fils qui n'auraient pas été validés, pas tout les autres avant.

Afin de simplifier l'utilisation de l'architecture au maximum, nous avons créé les fonctions suivantes:

```
/* ajoute le node en queue de la liste chainee de fils de n */
void addChild(Node *n, char *name);
void deleteChildren(Node *n);
void deleteChildrenFromIndex(Node *n, int k);
/* retourne la fonction de validation du noeud passé en argument (en fonction de sa
rulename) */
int(* getValidationFunction(Node *n))(char **req, Node *n);
/* valide tous les fils du node n */
int validateChildren(char **req, Node *n);
/* valide tous les freres du node n (inclu) */
int validateBrothers(char **req, Node *n);
/* valide tous les fils du node n, a partir du start-ieme (avec 0 on valide tout
les fils) */
int validateChildrenStartingFrom(char **req, Node *n, int start);
```

Voici un exemple typique d'utilisation:

```
int validateHttpVersion(char **req, Node *n)
{
    n->start = *req;
    addChild(n, "HTTP-name");
    addChild(n, "/");
    addChild(n, "DIGIT");
    addChild(n, ".");
    addChild(n, "DIGIT");

    if (!validateChildren(req, n))
    {
        deleteChildren(n);
        return 0;
    }
    return 1;
}
```

`**req` désigne le curseur sur la requête à parser, et `n` le noeud que nous cherchons à valider.

La fonction addChild complète la liste chainée de fils du Node n. Ainsi pour valider un noeud, on ajoute tout ses fils, qui seront validés par la méthode `validateChildren`.

De cette façon, la validation d'un noeud est juste la traduction en code de la grammaire. Toute la gestion des listes chainées, des attributs des noeuds, est cachée dans `validateChildren`.