Petit mémento pour étendre ILP

1 ILP1

Voici les points où intervenir en ILP1 si l'on souhaite l'étendre avec la caractéristique *hux*.

1. S'il y a création ou modification syntaxique, créer `Grammars/grammar1hux.rnc` (qui probablement inclut `grammar1.rnc`)
2. écrire quelques programmes de test (de nom suffixé par `-1hux.xml`) valides pour cette grammaire, ces programmes seront placés dans `Grammars/Samples/`. Écrire aussi le résultat (suffixe `.result`) et les impressions (suffixe `.print`) attendus.
3. écrire l'analyseur syntaxique pour traiter l'extension *hux* :
   (a) créer un nouveau paquetage `fr.upmc.ilp.ilp1hux` pour contenir les adjonctions à écrire en Java
   (b) créer une interface décrivant les nouveaux noeuds de l'AST dans `fr.upmc.ilp.ilp1hux.interfaces`.
   (c) implanter cette interface dans `fr.upmc.ilp.ilp1hux.ast`.
   (d) écrire l'interface décrivant la nouvelle fabrique (probablement par héritage de `fr.upmc.ilp.ilp1.eval.IEASTFactory`)
   (e) écrire la fabrique `fr.upmc.ilp.ilp1hux.ast.Factory` (probablement par héritage de `fr.upmc.ilp.ilp1.eval.EASTFactory`)
   (f) enfin écrire l'analyseur `fr.upmc.ilp.ilp1hux.ast.Parser` (probablement par héritage de `fr.upmc.ilp.ilp1.eval.EASTParser`)
4. écrire les méthodes nécessaires pour l'interprétation (et notamment `eval`) dans les classes implantant les nouveaux noeuds d'AST en `fr.upmc.ilp.ilp1hux.ast`.
5. ainsi qu'éventuellement les méthodes ou valeurs supplémentaires dans la bibliothèque d'exécution (en Java) dans `fr.upmc.ilp.ilp1hux.runtime`.
6. modifier éventuellement les implantations des environnements global ou local
7. décrire comment traiter (préparer, interpréter) le nouveau langage avec la classe `fr.upmc.ilp.ilp1hux.Process`
8. décrire comment tester la préparation et l'interprétation de ce nouveau langage avec la classe `fr.upmc.ilp.ilp1hux.ProcessTest`
9. écrire les méthodes nécessaires pour la compilation vers C dans `fr.upmc.ilp.ilp1hux.cgen.` : raffiner la méthode `analyze` et écrire les méthodes `generate` pour chaque nouveau noeud d'AST
10. ainsi qu'éventuellement les fonctions ou valeurs supplémentaires dans la bibliothèque d'exécution (en C) dans `C/`
11. enrichir `fr.upmc.ilp.ilp1hux.Process` pour compiler le nouveau langage
12. enrichir `fr.upmc.ilp.ilp1hux.ProcessTest` pour tester la compilation de ce nouveau langage

2 ILP2

Voici les points où intervenir en ILP2 si l'on souhaite l'étendre avec la caractéristique *hux*. ILP3 est un exemple d'extension d'ILP2.

1. S'il y a création ou modification syntaxique, créer `Grammars/grammar2hux.rnc` (qui probablement inclut `grammar2.rnc`)
2. écrire quelques programmes de test (de nom suffixé par `-2hux.xml`) valides pour cette grammaire, ces programmes seront placés dans `Grammars/Samples/`. Écrire aussi le résultat (suffixe `.result`) et les impressions (suffixe `.print`) attendus.
3. créer un nouveau paquetage `fr.upmc.ilp.ilp2hux` pour contenir les adjonctions à écrire en Java. S'il y a peu de classes, pas la peine de créer les sous-paquetages `interfaces`, `ast`, `runtime`, etc.
4. créer, si nécessaire, les interfaces décrivant les nouveaux noeuds de l'AST dans `fr.upmc.ilp.ilp2hux.interfaces.IAST2*`
5. implanter ces interfaces avec les classes `fr.upmc.ilp.ilp2hux.ast.CEAST*` en héritant de `CEASTinstruction` ou `CEASTexpression`. Chacune de ces classes comporte :
   – la méthode d'analyse syntaxique (méthode statique `parse`),
   – la méthode d'interprétation (`eval`),
   – écrire éventuellement les méthodes ou valeurs supplémentaires dans la bibliothèque d'exécution (en Java) dans `fr.upmc.ilp.ilp2hux.runtime`.
   – modifier éventuellement les implantations des environnements global ou local
   – la méthode de compilation (`compileInstruction` ou `compileExpression`),
   – ainsi qu'éventuellement les fonctions ou valeurs supplémentaires dans la bibliothèque d'exécution (en C) dans `C/`
   – la méthode de recherche des variables libres (`findFreeVariables`)
6. enfin écrire l'analyseur `fr.upmc.ilp.ilp2hux.ast.Parser` (probablement par héritage de `fr.upmc.ilp.ilp2.ast.CEASTParser`) invoquant les méthodes `parse` écrites plus haut

7. éventuellement, écrire la classe `fr.upmc.ilp.ilp2hux.ast.CEASTprogram` (probablement par héritage de `fr.upmc.ilp.ilp2.ast.CEASTprogram`)
8. décrire comment traiter (préparer, interpréter) le nouveau langage avec la classe `fr.upmc.ilp.ilp2hux.Process`
9. décrire comment tester la préparation et l'interprétation de ce nouveau langage avec la classe `fr.upmc.ilp.ilp2hux.ProcessTest`

3 ILP4

Voici les points où intervenir en ILP4 si l'on souhaite l'étendre avec la caractéristique *hux*. ILP6 est un exemple (complexe) d'extension d'ILP4.

1. S'il y a création ou modification syntaxique, créer `Grammars/grammar4hux.rnc` (qui probablement inclut `grammar4.rnc`)
2. écrire quelques programmes de test (de nom suffixé par `-4hux.xml`) valides pour cette grammaire, ces programmes seront placés dans `Grammars/Samples/`. Écrire aussi le résultat (suffixe `.result`) et les impressions (suffixe `.print`) attendus.
3. créer un nouveau paquetage `fr.upmc.ilp.ilp4hux` pour contenir les adjonctions à écrire en Java. S'il y a peu de classes, pas la peine de créer les sous-paquetages `interfaces`, `ast`, `runtime`, etc.
4. créer, si nécessaire, les interfaces décrivant les nouveaux noeuds de l'AST dans `fr.upmc.ilp.ilp4hux.interfaces.IAST4*`
5. implanter ces interfaces avec les classes `fr.upmc.ilp.ilp4hux.ast.CEAST*` en héritant de `CEASTinstruction` ou `CEASTexpression`. Chacune de ces classes comporte :
   - la méthode d'analyse syntaxique (méthode statique `parse`),
   - les annotations (`@ILPexpression` ou `@ILPvariable`) sur les méthodes (les accesseurs) menant à des sous-expressions ou à des variables
   - la méthode d'interprétation (`eval`),
   - écrire éventuellement les méthodes ou valeurs supplémentaires dans la bibliothèque d'exécution (en Java) dans `fr.upmc.ilp.ilp4hux.runtime`.
   - modifier éventuellement les implantations des environnements global ou local
   - la méthode de normalisation (`normalize`)
   - la méthode déterminant les fonctions invoquées (`findInvokedFunctions`)
   - la méthode d'intégration des appels (`inline`)
   - la méthode de compilation (`compile`),
   - ainsi qu'éventuellement les fonctions ou valeurs supplémentaires dans la bibliothèque d'exécution (en C) dans `C/`
   - la méthode technique de visite (`accept`).
6. enfin configurer (avec `addParser`) l'analyseur instance de `fr.upmc.ilp.ilp4.ast.CEASTParser`) invoquant les méthodes `parse` écrites plus haut
7. écrire la classe `fr.upmc.ilp.ilp4hux.ast.CEASTprogram` (probablement par héritage de `fr.upmc.ilp.ilp4.ast.CEASTprogram`)
8. décrire comment traiter (préparer, interpréter) le nouveau langage avec la classe `fr.upmc.ilp.ilp4hux.Process`
9. décrire comment tester la préparation et l'interprétation de ce nouveau langage avec la classe `fr.upmc.ilp.ilp4hux.ProcessTest`