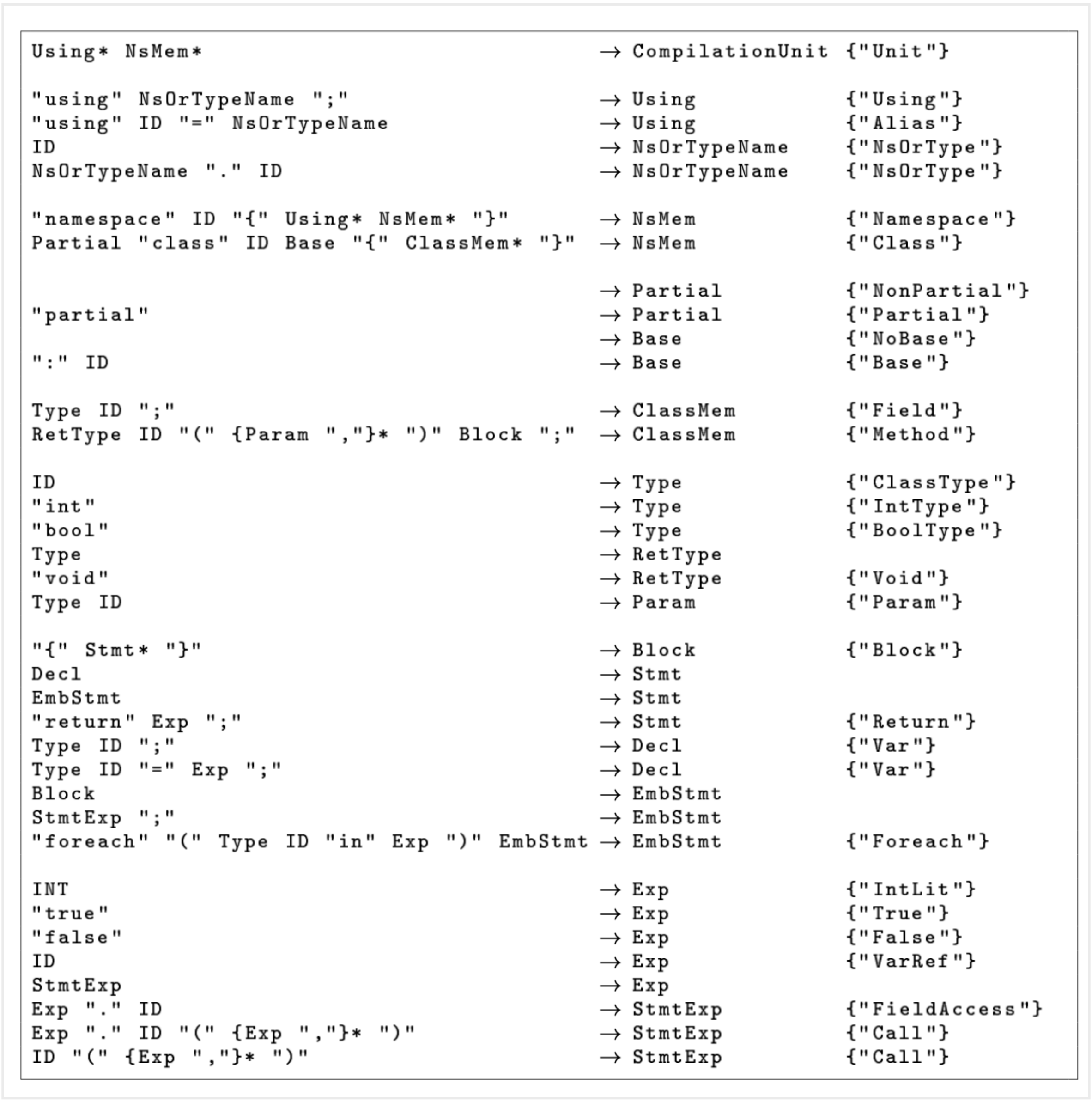
Declarative Name Binding and Scope Rules

Name binding is concerned with the relation between definitions and references of identifiers in textual software languages, including scope rules that govern these relations.

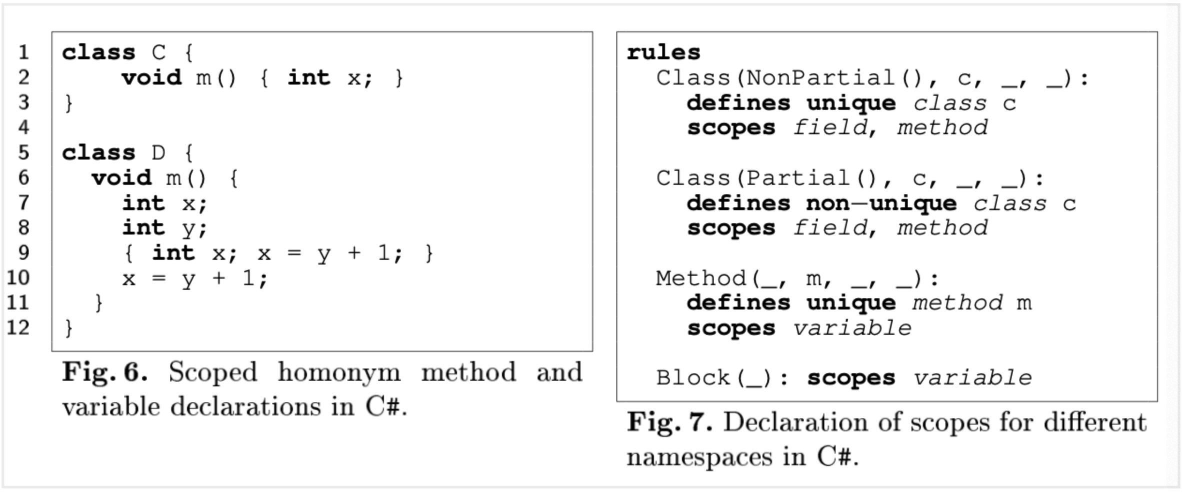


```
Using* NsMem*                                    → CompilationUnit {"Unit"}

"using" NsOrTypeName ";"                         → Using           {"Using"}
"using" ID "=" NsOrTypeName                      → Using           {"Alias"}
ID                                               → NsOrTypeName    {"NsOrType"}
NsOrTypeName "." ID                              → NsOrTypeName    {"NsOrType"}

"namespace" ID "{" Using* NsMem* "}"             → NsMem           {"Namespace"}
Partial "class" ID Base "{" ClassMem* "}"        → NsMem           {"Class"}

                                                 → Partial         {"NonPartial"}
"partial"                                        → Partial         {"Partial"}
                                                 → Base            {"NoBase"}
":" ID                                           → Base            {"Base"}

Type ID ";"                                      → ClassMem        {"Field"}
RetType ID "(" {Param ","}* ")" Block ";"        → ClassMem        {"Method"}

ID                                               → Type            {"ClassType"}
"int"                                            → Type            {"IntType"}
"bool"                                           → Type            {"BoolType"}
Type                                             → RetType
"void"                                           → RetType         {"Void"}
Type ID                                          → Param           {"Param"}

"{" Stmt* "}"                                    → Block           {"Block"}
Decl                                             → Stmt
EmbStmt                                          → Stmt
"return" Exp ";"                                 → Stmt            {"Return"}
Type ID ";"                                      → Decl            {"Var"}
Type ID "=" Exp ";"                              → Decl            {"Var"}
Block                                            → EmbStmt
StmtExp ";"                                      → EmbStmt
"foreach" "(" Type ID "in" Exp ")" EmbStmt       → EmbStmt         {"Foreach"}

INT                                              → Exp             {"IntLit"}
"true"                                           → Exp             {"True"}
"false"                                          → Exp             {"False"}
ID                                               → Exp             {"VarRef"}
StmtExp                                          → Exp
Exp "." ID                                       → StmtExp         {"FieldAccess"}
Exp "." ID "(" {Exp ","}* ")"                    → StmtExp         {"Call"}
ID "(" {Exp ","}* ")"                            → StmtExp         {"Call"}
```

<<<Esta linguagem parece a melhor para usar na minha tese>>>

Scopes

Scopes restrict the visibility of definition sites. A named scope is the definition site for a name which scopes other definition sites. By contrast, an anonymous scope does not define a name. Scopes can be nested and name resolution typically looks for definition sites from inner to outer scopes.



```
class C {
    void m() { int x; }
}

class D {
  void m() {
    int x;
    int y;
    { int x; x = y + 1; }
    x = y + 1;
  }
}
```

Fig. 6. Scoped homonym method and variable declarations in C#.

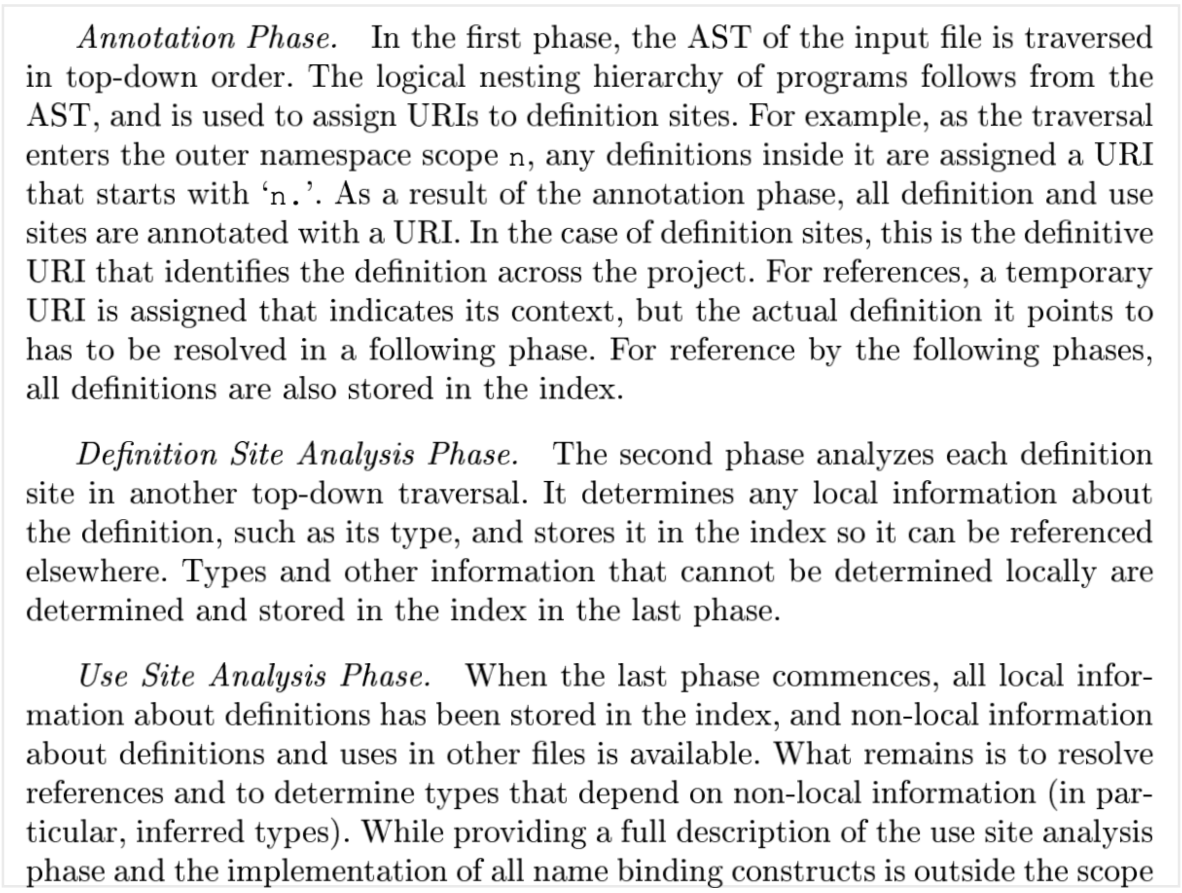
```
rules
  Class(NonPartial(), c, _, _):
    defines unique class c
    scopes field, method

  Class(Partial(), c, _, _):
    defines non−unique class c
    scopes field, method

  Method(_, m, _, _):
    defines unique method m
    scopes variable

  Block(_): scopes variable
```

Fig. 7. Declaration of scopes for different namespaces in C#.

Imports

An import introduces into the current scope definitions from another scope, either under the same name or under a new name. An import that imports all definitions can be transitive.

Resolving Names Algorithm



Annotation Phase. In the first phase, the AST of the input file is traversed in top-down order. The logical nesting hierarchy of programs follows from the AST, and is used to assign URIs to definition sites. For example, as the traversal enters the outer namespace scope `n`, any definitions inside it are assigned a URI that starts with '`n.`'. As a result of the annotation phase, all definition and use sites are annotated with a URI. In the case of definition sites, this is the definitive URI that identifies the definition across the project. For references, a temporary URI is assigned that indicates its context, but the actual definition it points to has to be resolved in a following phase. For reference by the following phases, all definitions are also stored in the index.

Definition Site Analysis Phase. The second phase analyzes each definition site in another top-down traversal. It determines any local information about the definition, such as its type, and stores it in the index so it can be referenced elsewhere. Types and other information that cannot be determined locally are determined and stored in the index in the last phase.

Use Site Analysis Phase. When the last phase commences, all local information about definitions has been stored in the index, and non-local information about definitions and uses in other files is available. What remains is to resolve references and to determine types that depend on non-local information (in particular, inferred types). While providing a full description of the use site analysis phase and the implementation of all name binding constructs is outside the scope

Resumo

Este artigo foca-se mais em especificar a "Spoofax Naming Binding Language" e depois em verificar name bindings e scopes.

Esta linguagem parece-me a mais apropriada para usar porque é baseada em C# e como é orientada aos objetos penso que se aproxime mais com o âmbito da minha tese, visto que esse paradigma é o mais intricado no que toca a scope rules.
O artigo fala também sobre a utilidade deste tipo de ferramentas (integração em IDEs) e de como é implementada (Index API).
Finalmente falam sobre as limitações e coverage desta framework.
Já percebi que o problema deste projeto é mesmo fazer a ferramenta de modo a que a abstração cubra a generalidade das linguagens…