Chapter 1

ADL

In this chapter we define the structure, the syntax and the semantics of ADL. As abbreviation, ADL stands for A Description Language. This specification can be used to define a repository in which to store rules in their contexts.

1.1 Concepts

Concepts are treated as sets. You may think of the concept `Person`, for example, as a set in which you might find the elements `Abraham`, `Bob`, and `Charlotte`. As a consequence, all well known properties of sets are valid for concepts as well. For instance, if a concept C has element `Bob` and $C \subseteq C'$, then `Bob` is an element of C' also. We will use the word *atom* to discuss elements of concepts, indicating that these elements are the smallest things we consider in this chapter. The collection of all conceivable atoms is sometimes called 'universe of discourse' in the literature. We talk about the *type* of an atom as in `Abraham`, `Bob`, and `Charlotte` have type `Person`. The set of atoms that corresponds to a concept is called the *population* of that concept.

Since relations are (by definition) sets, a relation has elements too. For example, the link $(\texttt{Bush}, \texttt{USA})$ might be an element of a relation `president`. Two links are equal if their left atoms are equal and their right atoms are equal. All links in a relation have the same types. That is: the left atoms of all links in a relation have the same type, and so do the right atoms. The concept to the left is called *source* and the concept to the right is the *target*. Within a context, every relation is defined uniquely by its name and the source and target concepts. We write $R : A \times B$ to denote the relation with name R, source A and target B.

The symbol $\subseteq$ is called the 'isa' relation when applied to concepts. When applied to sets, the relation $\subseteq$ is called 'subset'. For instance `Judge` $\subseteq$ `Person` means that the concept `Judge` is more specific than the concept `Person`. The set of judges (corresponding to `Judge`) is a subset of the set of persons (which corresponds to `Person`).

1.2 Rules

ADL uses the idea of patterns to collect a number of rules that describe one particular theme[1]. So rules are defined within a *pattern*, which is therefore a collection of rules. When that pattern is used in a context, all rules defined in that pattern apply within the context. Within the context itself, extra rules may be defined for the purpose of glueing patterns together[2]. So, all rules that apply in a context are the ones defined in patterns used by the context plus the rules defined within that context.

The syntax of rules is given in Backus Naur Form (BNF):

```
Rule ::= Expression "-:" Expression "." .
Rule ::= Morphism "=" Expression "." .
```

You can read the first line as: a rule can be an expression followed by the `-:` symbol, followed by another expression and terminated by a dot (full stop symbol). An example of a rule is `elem;subset -: elem`, in which `elem;subset` is an expression and `elem` is an expression too. Similarly, you read the second line as: a rule can be a morphism followed by the `=` symbol, followed by an expression and terminated by a dot.

The syntax of expressions is given by:

```
Expression ::= Morphism .
Expression ::= Expression "~" .
Expression ::= Expression "-" .
Expression ::= Expression ";" Expression .
Expression ::= Expression "/\" Expression .
Expression ::= Expression "\/" Expression .
Expression ::= "(" Expression ")" .
```

[1]Patterns are discussed in section 1.5

[2]Glueing of patterns is discussed in ??

A morphism is part of a rule that denotes one relation in any context where this rule applies. For instance, the rule `elem;subset -: elem` contains morphisms `elem` and `subset`. Usually, the name alone is sufficient to identify a relation uniquely. In some cases it is necessary to write the full signature (for example `elem[Atom*Set]`, i.e. the name together with source- and target concepts) to prevent ambiguity. If ADL can deduce the source and target concepts, however, it is sufficient to write a morphism just by mentioning its name (the identifier). Identifiers of a morphism always start with a lower case letter (`mIdent`), but names that identify concepts always start with a capital letter (`cIdent`). The syntax of morphisms is given by:

```
  Morphism ::= mIdent .
  Morphism ::= mIdent "[" cIdent "*" cIdent "]" .
  Morphism ::= mIdent "[" cIdent "]" .
```

If you specify the full signature, as in `president[Name*Country]`, you know for sure that this is the signature used by ADL. If you specify only one concept, as in `friend[Name]`, ADL interprets this as `friend[Name*Name]`. If you do not specify any concepts, ADL will try to find out which signature you mean. When that is unique (which is often the case), you do not have to specify source and target.

You always work in one particular context, called the *current context.* Every morphism is bound to precisely one relation in your current context. Notice that the same morphism may be bound to different relations in different contexts, because one rule (which is defined in a pattern) applies in all contexts that use this rule.

If you work in a context (e.g. the context of Marlays bank) you may define a new context (e.g. Mortgages) as an extention of an existing context. This means that all rules that apply in the context 'Marlays bank' apply in the context 'Mortgages' as well. The rules that apply in the generic context ('Marlays bank') are a subset of the rules that apply in the specific context ('Mortgages').

1.3 Relations

Any relation is a set of links. For example:

```
  president = { ("Bush","USA"),
                ("Clinton","USA"),
                ("Chirac","France") }
```

Each relation has a signature too, for instance:

```
  president :: Name * Country
```

The concept on the left hand side is called the *source* and the right hand side concept is called *target*.

Every link has a left atom and a right atom. The type of the left atom Within any context, the signature (i.e. name, source and target) determines a relation uniquely.

The left atom of a link is in the set that corresponds to the source concept of that link.

The right atom of a link is in the set that corresponds to the target concept of that link.

A tuple in a relation matches the signature of that relation. That is: the left atom of a tuple is atom of the source of the relation in which that tuple resides. Idem for the target.

Any link in relation r is also in relations of which r is a subrelation. The reason is that a relation is a set of links, so subsets are subrelations.

1.4 Valuations

Any rule is a statement that can be applied to many different atoms. For example, the rule:

```
  president -: citizen .
```

This rule says specifies that the relation `president` is a subset of the relation `citizen`. So it says that you can be a president only if you are a citizen. We can apply this rule to different atoms, to see whether it is true or not. For instance, let

```
  president = {("Bush","USA"), ("Clinton","USA"),
               ("Chirac","France")}
  citizen   = {("Bush","USA"), ("Clinton","USA"),
               ("Chirac","France"), ("Joosten","Netherlands")}
```

In this example, the rule is obviously true. For every value we choose, we get the expected true or false statement: e.g.

"Bush" president "USA" implies
that "Bush" citizen "USA".

is clearly true. On the other hand:

"Joosten" president "USA" implies
that "Chirac" citizen "France".

is clearly false.

In order to discuss semantics of rules precisely, we introduce the notion of *valuation.* A valuation is one particular assignment of atoms to a rule that makes the rule true or false. For that purpuse, a valuation needs to allocate a value (i.e. a link) to each morphism in the rule. In the example we have two morphisms (`president` and `citizen`), so a valuation gives values to each one. In the first example the following valuation was used:

```
president -> ("Bush","USA"); citizen -> ("Bush","USA")
```

The second example used a different valuation:

```
president->("Joosten","USA"); citizen->("Chirac","France")
```

For every valuation of rule r that contains a link l, this link is an element of a relation in each context in which r applies.

1.5 Patterns

Every signature used in a rule is defined in the same pattern as that rule.

A relation is bound to a signature, which is defined in a pattern used in the relation's context.

Any relation in a context is also known in more generic contexts. The reason is that a relation is a set of links, so subsets are subrelations.

A pattern used by a context is implicitly used by more specific contexts.

If one relation is a subrelation of another one (the super-relation), it means that they have compatible signatures and the subrelation is in the same or a more specific context than the super-relation.

Chapter 2

Detailed descriptions

This chapter defines ADL in a formal way[1]. Any implementation that satisfies the properties defined in this chapter is a valid implementation of ADL. We will specify ADL in terms of sets, assuming that you are familiar with sets, especially with the relations $\in$ (element of) and $\subseteq$ (subset).

2.1 Concepts

A concept is a collection of things of one kind. The relation *isa* expresses that one concept may be more specific than another. This relation is familiar to object-oriented programmers and artificial intelligence professionals.

$$isa :: \mathit{Concept} \leftrightarrow \mathit{Concept}$$

For instance: `Teacher` *isa* `Person` means that `Teacher` is a more specific concept than `Person`. Just like $\subseteq$, *isa* is reflexive, transitive, and antisymmetric. The relation *isa* applies to concepts, whereas $\subseteq$ applies to sets.

The correspondence between concepts and sets is obtained by the function *pop*, which associates a set to every concept.

$$pop :: \mathit{Concept} \rightarrow \mathit{Set}$$

The relation *isa* must satisfy the following property, for all concepts c and c':

$$c\ isa\ c' \ \Rightarrow\ pop(c) \subseteq pop(c') \tag{2.1}$$

This property says for instance that `Teacher` *isa* `Person` implies that the set of teachers (corresponding to `Teacher`) is a subset of the set of persons

[1]The entire specification was conceived with the help of the ADL language itself.

(which corresponds to `Person`). That is: every atom in the set of teachers is also a member of the set of persons.

For any atom a that is element of a set that corresponds to a concept c, we say that a has type c. We use the following relation.

$$\mathit{type} :: \mathit{Atom} \leftrightarrow \mathit{Concept}$$

This relation is total, since every atom has a type. A type is a concept in whose population the atom occurs. It is defined by:

$$\forall\, a :: \mathit{Atom};\; c :: \mathit{Concept} : a\ \mathit{type}\ c \;\Leftrightarrow\; a \;\in\; \mathit{pop}(c) \tag{2.2}$$

A combination of two atoms that occurs in a relation is called a *link*. One atom is called the *left atom* and the other one is the *right atom*. The functions *left* and *right* yield the respective atoms.

$$\mathit{left}, \mathit{right} :: \mathit{Link} \rightarrow \mathit{Atom}$$

Each link is fully determined by its left and right atoms.

$$\forall\, l, l' :: \mathit{Link} : l = l' \;\Leftrightarrow\; \mathit{right}(l) = \mathit{right}(l') \;\wedge\; \mathit{left}(l) = \mathit{left}(l') \tag{2.3}$$

When two atoms are linked in a relation, a link implicitly links two concepts. The left concept is called the *source* and the right concept *target*. Functions *src* and *trg* map a link to either concept.

$$\mathit{src}, \mathit{trg} :: \mathit{Link} \rightarrow \mathit{Concept}$$

These functions are defined by:

$$\forall\, l :: \mathit{Link} :\; \mathit{src}(l) = \mathit{type}(\mathit{left}(l)) \;\wedge\; \mathit{trg}(l) = \mathit{type}(\mathit{right}(l)) \tag{2.4}$$

Figure 2.1 illustrates the structure of the previously defined rules. This diagram shows an arrow for every function and a line for every relation. The tiny arrow halfway a line determines the order of arguments in each relation.

The following fragment represents the specification in terms of ADL:

```
PATTERN Concepts
 elem   :: Atom * Set.
 subset :: Set * Set [RFX,TRN,ASY].
 isa    :: Concept * Concept [RFX,TRN,ASY].
 pop    :: Concept * Set [UNI,TOT].
 isa    -: pop ; subset ; pop˜.
```

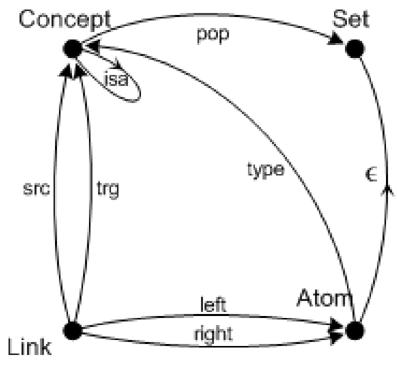

Figure 2.1: Structure of Concepts

```
 type   :: Atom * Concept [TOT].
 type    = elem ; pop~.
 left   :: Link * Atom [UNI,TOT].
 right  :: Link * Atom [UNI,TOT].
 I[Link] = right ; right~ /\ left ; left~.
 src    :: Link*Concept [UNI,TOT].
 trg    :: Link*Concept [UNI,TOT].
 left   -: src ; pop ; elem~.
 right  -: trg ; pop ; elem~.
ENDPATTERN
```

2.2 Rules

Rules are the most prominent feature of ADL. They are defined in patterns and a pattern is used by any number of contexts. Which rule is defined in which pattern is represented by the function *definedIn*.

$$definedIn :: Rule \rightarrow Pattern$$

Which context uses which pattern is given by the relation *uses*:

$$uses :: Context \leftrightarrow Pattern$$

When a pattern is used in a context, all rules of that pattern apply within that context. So, we introduce a third relation, *appliesIn*, to express which rules apply in which context:

$$appliesIn :: Rule \leftrightarrow Context$$

These relations satisfy equation 2.5, which states that for each rule r and every context c:

$$r \ \mathit{appliesIn} \ c \ \Leftrightarrow \ c \ \mathit{uses} \ \mathit{definedIn}(r) \tag{2.5}$$

Within the context itself, extra rules may be defined for the purpose of glueing patterns together. So all rules that apply in a context are the ones defined in patterns used by the context plus the rules defined within that context.

New contexts can be defined as extensions of existing contexts. This is represented by the relation *extends*.

$\mathit{extends} :: \mathit{Context} \leftrightarrow \mathit{Context}$

This relation is reflexive, transitive, and antisymmetric. If you work in a context (e.g. the context of Marlays bank) you may define a new context (e.g. Mortgages) as an extension of an existing context. This means that all rules that apply in the context `Marlays` apply in the context `Mortgages` as well. The rules that apply in the generic context (`Marlays`) are a subset of the rules that apply in the specific context (`Mortgages`). This property is defined for every rule r and all contexts c and c' by equation 2.6.

$$r \ \mathit{appliesIn} \ c' \ \wedge \ c \ \mathit{extends} \ c' \ \Rightarrow \ r \ \mathit{appliesIn} \ c \tag{2.6}$$

Relations are defined in a context. This information is available by means of the function *in*:

$\mathit{in} :: \mathit{Relation} \rightarrow \mathit{Context}$

If, for example (‘Relation 1’, ‘RAP’) occurs in relation *in*, this means that relation ‘Relation 1’ is available in context ‘RAP’.

A *morphism* is part of a rule that denotes a relation in the context of this rule. The relation *in* tells us which morphisms are used in which rule.

$\mathit{in} :: \mathit{Morphism} \leftrightarrow \mathit{Rule}$

This relation is surjective and total, since every morphism is used in at least one rule and every rule contains at least one morphism. The signature of a morphism is given by the function *sign*:

$\mathit{sign} :: \mathit{Morphism} \rightarrow \mathit{Signature}$

Likewise, the signature of a morphism is given by another function *sign*:

$\mathit{sign} :: \mathit{Relation} \rightarrow \mathit{Signature}$

This allows us to express in property 2.7 that every morphism in a rule is bound to a relation in the applicable context. For every rule r, context c, and morphism m

$$\begin{array}{l} m \ in \ r \ \wedge \ r \ appliesIn \ c \ \Rightarrow \\ \exists \, r' :: Relation : \ sign(m) = sign(r') \ \wedge \ in(r') = c \end{array} \tag{2.7}$$

You always work in one particular context, called the *current* context. Every morphism is bound to precisely one relation in the current context[2].

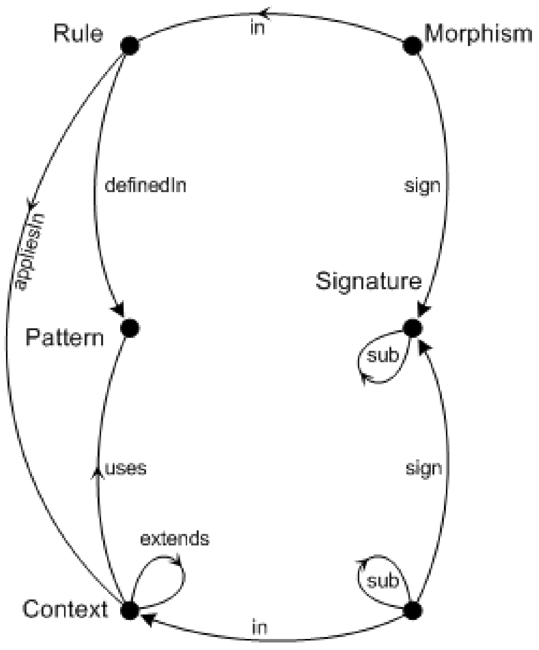

Figure 2.2: Structure of Rules

The following fragment represents the specification in terms of ADL:

```
PATTERN Rules
 definedIn :: Rule * Pattern [UNI,TOT].
 uses      :: Context * Pattern.
 appliesIn :: Rule * Context.
 appliesIn  = definedIn;uses˜.
 extends   :: Context * Context [RFX,TRN,ASY].
 appliesIn ; extends˜ -: appliesIn.
 sign      :: Morphism * Signature [UNI,TOT].
 in        :: Morphism * Rule [SUR,TOT].
 sign      :: Relation * Signature [UNI,TOT].
```

[2]TBD: insert proof here

```
 in        :: Relation * Context [UNI,TOT].
 in ; appliesIn -: sign ; sign~ ; in.
ENDPATTERN
```

2.3 Relations

Within any context, the signature (i.e. name, source and target) determines a relation uniquely. All relations r and s satisfy property 2.8.

$$r = s \;\Leftrightarrow\; \mathit{sign}(r) = \mathit{sign}(s) \wedge \mathit{in}(r) = \mathit{in}(s) \tag{2.8}$$

The definition of *in* and *sign* are given in section 2.2 (pg. 9). We introduce the notion of *subrelation* to express that any link in a subrelation of r is also in r itself. Since a relation is a set of links, this corresponds to the subset relation. For this purpose we introduce the relation *sub*.

$\mathit{sub} :: \mathit{Relation} \leftrightarrow \mathit{Relation}$

This relation is reflexive, transitive, and antisymmetric. It satisfies property 2.9 for every link l and all relations r and s:

$$l \in r \wedge s \;\mathit{sub}\; r \;\Rightarrow\; l \in s \tag{2.9}$$

Note that this property has the following consequence. For all relations s and r:

$$s \;\mathit{sub}\; r \;\Rightarrow\; s \subseteq r \tag{2.10}$$

Since a relation is a set, we use the notation $l \in r$ to express that a link l is an element of relation r. Every relation has a *signature*, being the name together with the source- and target concepts of that relation. The following functions are used to retrieve the individual components of a signature:

$\mathit{source}, \mathit{target} :: \mathit{Signature} \rightarrow \mathit{Concept}$
$\mathit{name} :: \mathit{Signature} \rightarrow \mathit{Identifier}$

A name, source and target identify a signature uniquely, which is expressed for all signatures s and s' by equation 2.11:

$$s = s' \;\Leftrightarrow\; \begin{array}{r} \mathit{name}(s){=}\mathit{name}(s') \;\;\wedge \\ \mathit{source}(s){=}\mathit{source}(s') \;\;\wedge \\ \mathit{target}(s){=}\mathit{target}(s') \end{array} \tag{2.11}$$

Every tuple in a relation must match the signature of that relation. That is: the type of the left atom of a tuple must be compatible with the source of the relation in which that tuple resides. Idem for the target. This rule states that you cannot put just anything in a relation, but the types of atoms must respect the signature of the relation. For every link l and relation r property 2.12 must hold.

$$l \in r \;\Rightarrow\; src(l) = source(sign(r)) \;\wedge\; trg(l) = target(sign(r)) \qquad (2.12)$$

Relations *src* and *trg* are discussed in section 2.1 (pg. 7).

One signature may be more generic than another, which is defined in the relation

$$sub :: Signature \leftrightarrow Signature$$

Like before, this relation is reflexive, transitive, and antisymmetric.

If one relation s is a subrelation of another relation r, both must have compatible signatures and s must be in the same or a more specific context than r. This requirement is given by equation 2.13

$$s\ sub\ r \;\Leftrightarrow\; sign(s)\ sub\ sign(r) \wedge in(s)\ extends\ in(r) \qquad (2.13)$$

The relations *sign*, *in*, and *extends* are discussed in section 2.2.

The structure of these rules is given in figure 2.3.

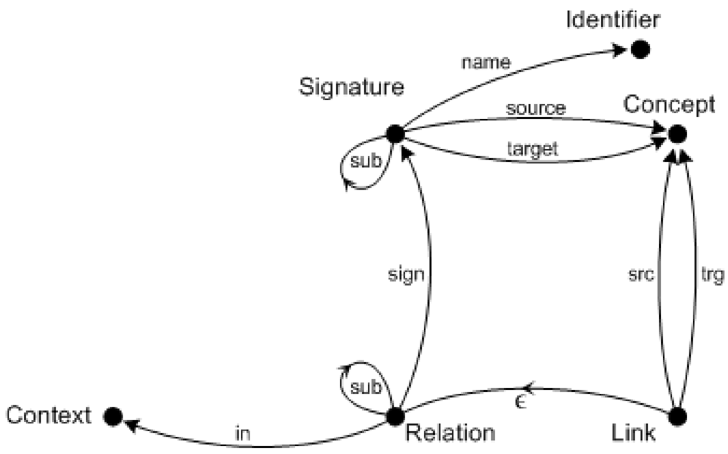

Figure 2.3: Structure of Relations

The following fragment represents the specification in terms of ADL:

```
PATTERN Relations
 source      :: Signature * Concept [UNI,TOT].
 target       :: Signature * Concept [UNI,TOT].
 in           :: Link * Relation.
 in ; sign   -: src ; ~source /\ trg ; ~target.
 I[Relation]  = sign;~sign /\ in;~in.
 sub          :: Relation * Relation [RFX,TRN,ASY].
 in ; sub    -: in.
 name         :: Signature * Identifier [UNI,TOT].
 I[Signature] = name; ~name /\
                source; ~source /\
                target; ~target.
 sub          :: Signature * Signature [RFX,TRN,ASY].
 sub           = sign;sub;~sign /\ in;extends;~in.
ENDPATTERN
```

2.4 Valuations

In order to discuss semantics of rules precisely, we introduce the notion of *valuation*. A valuation is one particular assignment of atoms to a rule that makes the rule true or false (see also section 1.4). For this purpuse, a valuation needs to allocate a value (i.e. a link) to each morphism in the rule.

$$\mathit{val} :: \mathit{Morphism} \leftrightarrow \mathit{Link}$$

All links in a valuation are registered in the relation *in*

$$\mathit{in} :: \mathit{Link} \leftrightarrow \mathit{Valuation}$$

The following definition relates rules to valuations.

$$\mathit{val} :: \mathit{Rule} \leftrightarrow \mathit{Valuation}$$

For each rule r, link l and morphism m, these relations must satisfy property 2.14.

$$m\ \mathit{in}\ r \wedge m\ \mathit{val}\ l\ \Rightarrow\ \exists\, v :: \mathit{Valuation}\ : r\ \mathit{val}\ v\ \wedge\ l\ \mathit{in}\ v \qquad (2.14)$$

The definition of $\mathit{in}[\mathit{Morphism} \times \mathit{Rule}]$ is given in section 2.2 (pg. 9).

For every valuation of rule r that contains a link l, and every valuation v, l is an element of a relation in each context c in which r applies.

$$\begin{array}{c} l\ \mathit{in}\ v \wedge r\ \mathit{val}\ v \wedge r\ \mathit{appliesIn}\ c\ \Rightarrow \\ \exists\, r' :: \mathit{Relation} : l \in\ r' \wedge \mathit{in}(r')\ =\ c \end{array} \qquad (2.15)$$

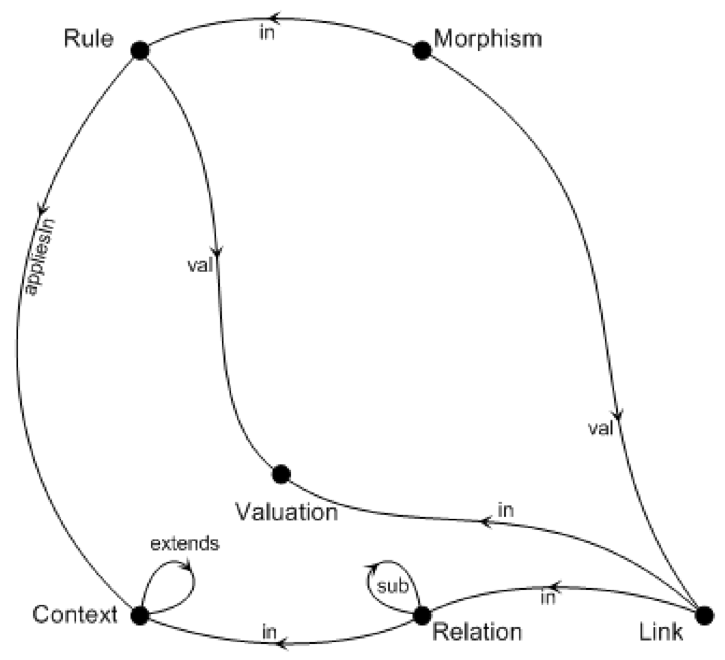

Figure 2.4: Structure of Valuations

The definition of *appliesIn* is given in section 2.2 (pg. 8) and *in* is discussed in section 2.2 (pg. 9).

The structure of these definitions is illustrated in figure 2.4.

The following fragment represents the specification in terms of ADL:

```
PATTERN Valuations
 val               :: Rule * Valuation.
 val               :: Morphism * Link.
 in                :: Link * Valuation.
 in~;val           -: val;in~.
 in;val~;appliesIn -: in;in.
ENDPATTERN
```

Valuations are used to define the semantics of ADL. We introduce the notation $\mathcal{C} \models a\ r\ b$ to denote that the value $a\ r\ b$ of a rule r is true in context $\mathcal{C}$.

$$\begin{array}{lll}
\mathcal{C} \models a\ \mathtt{m}\ b & \text{iff } \mathtt{m}\ \mathit{val}\ (a,b) & \wedge \\
 & \quad \exists\, r :: \mathit{Rule} : \quad \mathtt{m}\ \mathit{in}\ r\ \wedge\ r\ \mathit{appliesIn}\ \mathcal{C} & \wedge \\
 & \quad \exists\, r :: \mathit{Relation} : \quad (a,b)\ \mathit{in}\ r\ \wedge\ \mathit{in}(r) = \mathcal{C} & \\
\mathcal{C} \models a\ (\mathtt{r;s})\ b & \text{iff there exists } c \text{ such that } \mathcal{C} \models a\ r\ c\ \wedge\ \mathcal{C} \models c\ s\ b & \\
\mathcal{C} \models a\ (\mathtt{r/\backslash s})\ b & \text{iff } \mathcal{C} \models a\ \mathtt{r}\ b\ \wedge\ \mathcal{C} \models a\ \mathtt{s}\ b & \\
\mathcal{C} \models a\ (\mathtt{r\backslash/s})\ b & \text{iff } \mathcal{C} \models a\ \mathtt{r}\ b\ \vee\ \mathcal{C} \models a\ \mathtt{s}\ b & \\
\mathcal{C} \models a\ (\mathtt{r\tilde{}})\ b & \text{iff } \mathcal{C} \models b\ \mathtt{r}\ a & \\
\mathcal{C} \models a\ (\mathtt{r-})\ b & \text{iff } \mathcal{C} \not\models b\ \mathtt{r}\ a & \\
\mathcal{C} \models a\ \mathtt{I}\ b & \text{iff } \mathcal{C} \models a = b &
\end{array}$$

2.5 Patterns

A pattern is a collection of rules plus the signatures of all morphisms used in these rules. The function *definedIn* for rules is discussed in section 2.2 (page 8). Besides that one, we need the following function to convey which signatures are defined in which patterns:

$$\mathit{definedIn} :: \mathit{Signature} \rightarrow \mathit{Pattern}$$

The signature of every morphism m used in a rule r is defined in the same pattern as rule r:

$$m\ \mathit{in}\ r\ \Rightarrow\ \mathit{definedIn}(r) = \mathit{definedIn}(\mathit{sign}(m)) \tag{2.16}$$

The relations *in* and *sign* are discussed in section 2.2 (pg. 9).

A relation r is bound to a signature, which is defined in a pattern used in the relation's context.

$$\mathit{in}(r)\ \mathit{uses}\ \mathit{definedIn}(\mathit{sign}(r)) \tag{2.17}$$

The relations *in*, *sign*, and *uses* in this property are discussed in section 2.2.

Any relation r in a context c is also known in more generic contexts than c. The reason is that a relation is a set of links, so subsets are subrelations.

$$\mathit{in}(r)\ \mathit{extends}\ c\ \Rightarrow\ \mathit{in}(r)\ =\ c \tag{2.18}$$

The definition of *extends* is given in section 2.2 (pg. 9).

A pattern p used by a context c is implicitly used by more specific contexts c':

$$c'\ \mathit{extends}\ c\ \wedge\ c\ \mathit{uses}\ p\ \ \Rightarrow c'\ \mathit{uses}\ p \tag{2.19}$$

The structure of these definitions is illustrated in figure 2.4.

The following fragment represents the specification in terms of ADL:

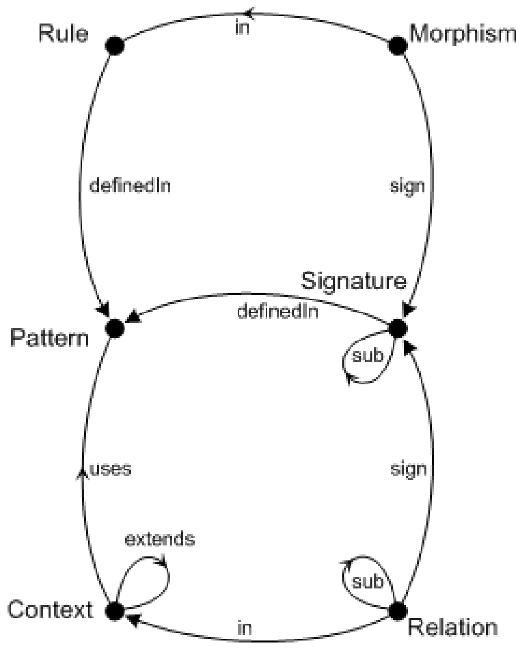

Figure 2.5: Structure of Patterns

```
PATTERN Patterns
 definedIn      :: Signature * Pattern [UNI,TOT].
 in˜;sign       -: definedIn; definedIn˜.
 in˜;sign       -: uses;definedIn˜
 in ; extends   -: in
 extends ; uses -: uses
ENDPATTERN
```

Chapter 3

Glossary

The structure of the complete specification is illustrated in figure 3.1.

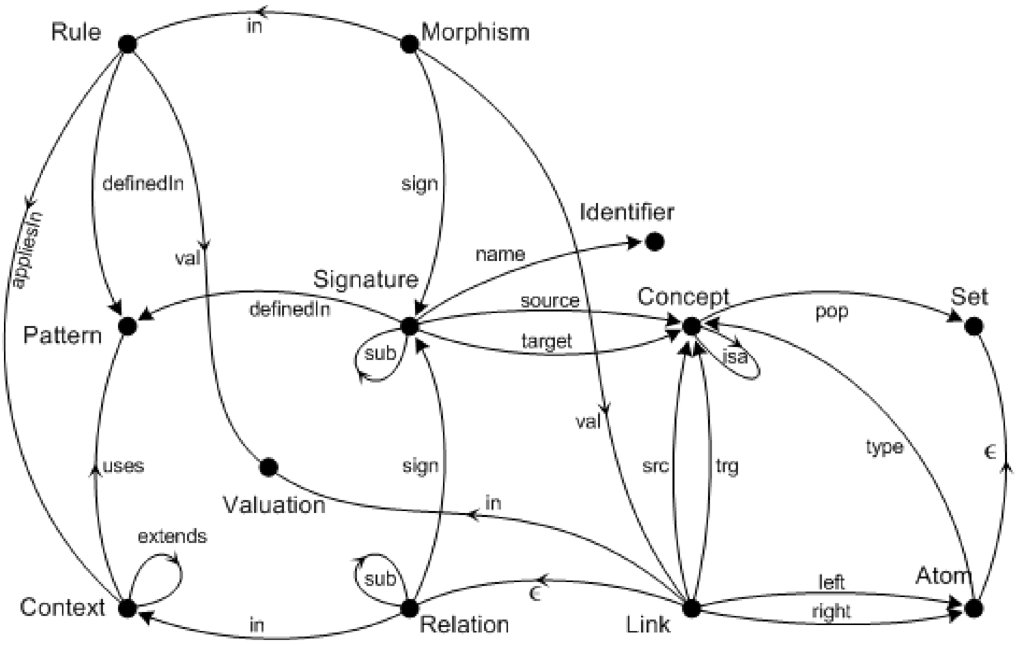

Figure 3.1: Entire Structure of ADL

The following concepts were used

Atom	Atomic object with one property only: it exists.
Concept	A collection of things of one kind. A concept corresponds to a set of atoms.

Context	A set of rules with power of law (within that context).
Identifier	A string that identifies a concept (if it starts with a capital letter) or a relation (if it starts with a lower case letter).
Link	The combination of two atoms, one to the left and one to the right.
Morphism	part of a rule that denotes one relation in any context where this rule applies. Usually, the name alone is sufficient to identify the relation uniquely. In some cases it is necessary to write the full signature (i.e. the name together with source- and target concepts) to prevent ambiguity.
Pattern	A set of rules inside an architecture that represents a generic solution to a design problem. A pattern has a name and can be used within a context. Patterns can be inserted into or removed from an architecture.
Relation	A subset of the cartesian product of a concept called its source (i.e. the left attribute) and a concept called its target (i.e. the right attribute).
Rule	An statement in relational algebra that restricts the possible interpretations of relations used in the rule.
Set	A collection of atoms.
Signature	The name, source- and target concepts of a relation or morphism.
Valuation	An assignment of links to every morphism in a relation. Every particular valuation makes a rule true or false within its context.