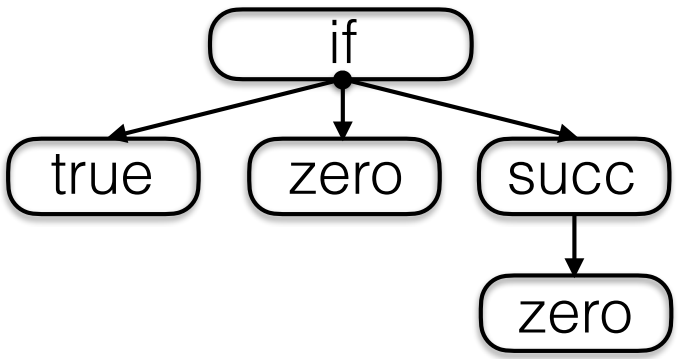
Tree-based abstract syntax

Ralf Lämmel
Software Language Engineering Team
University of Koblenz-Landau
http://www.softlang.org/

A conditional expression

Concrete syntax	***Abstract syntax***	
Text	**Tree**	**Tree as term**

if true
then zero
else succ zero
		`if(true,`
		`zero,`
		`succ(zero)).`

N.B.: abstract syntax is relevant in ***i)*** programming language theory as foundation for assigning semantics to programs and ***ii)*** metaprogramming for object program representation.

Signature-based definition of the abstract syntax of *expressions*

```
symbol true : → expr ; // The Boolean "true"
symbol false : → expr ; // The Boolean "false"
symbol zero : → expr ; // The natural number zero
symbol succ : expr → expr ; // Successor of a natural number
symbol pred : expr → expr ; // Predecessor of a natural number
symbol iszero : expr → expr ; // Test for a number to be zero
symbol if : expr × expr × expr → expr ; // Conditional
```

N.B.: A signature defines a set of terms.
For instance, the set defined by the signature at hand contains the following term:

```
if(true,
  zero,
  succ(zero)).
```

Conformance

- Term *t* is of sort *s* according to signature Σ, if:
  - Σ has a function symbol *f* such that:
    - *f* is the outermost symbol of *t*,
    - the result sort of *f* is *s*,
    - the arguments of *t* are terms of the argument sorts of *f* according to signature Σ.

Abstract syntax of *an imperative programming language*

```
// Statements
symbol skip : → stmt ;
symbol assign : string × expr → stmt ;
symbol seq : stmt × stmt → stmt ;
symbol if : expr × stmt × stmt → stmt ;
symbol while : expr × stmt → stmt ;

// Expressions
symbol intconst : integer → expr ;
symbol var : string → expr ;
symbol unary : uop × expr → expr ;
symbol binary : bop × expr × expr → expr ;
```

- string is a primitive type
- integer is a primitive type
- uop (unary symbols) elided
- bop (binary symbols) elided

Abstract syntax in the context of metaprogramming

- **<u>Meta</u>program** — program with programs as data
- **<u>Meta</u>language** — language in which metaprograms are written
- **<u>Object</u> program** — program manipulated by metaprogram
- **<u>Object</u> language** — language of object programs
- **<u>Object</u> program representation** — abstract syntax-based representation of object programs in metaprograms

N.B.: ‘object language’ should be understood in a broad sense to include programming languages as well as other kinds of software languages, e.g., modeling languages and exchange formats. Conformance may thus boil down to type checking or schema-based validation.

Object-program representation in **Haskell**

Algebraic data types

```
data Expr
  = TRUE | FALSE | Zero
  | Succ Expr | Pred Expr | IsZero Expr
  | If Expr Expr Expr
```

Term construction

```
sampleExpr :: Expr
sampleExpr =
  Pred (If (IsZero Zero) (Succ Zero) Zero)
```

Object-program representation in **Java**

```
public abstract class Expr { }
public class True extends Expr { }
public class False extends Expr { }
public class Zero extends Expr { }
public class Succ extends Expr {
    public Expr e;
    public Succ(Expr e) { this.e = e; }
}
public class Pred extends Expr { ... }
public class IsZero extends Expr { ... }
public class If extends Expr { ... }
```

Object model

```
Expr sample =
  new Pred(
    new If(
      new IsZero(new Zero()),
      new Succ(new Zero()),
      new Zero()));
```

Functional construction

Object-program representation in **JSON**

```
{
  "pred": {
    "if": {
      "x": { "iszero": { "zero": { } } },
      "y": { "succ": { "succ": { "zero": { } } } },
      "z": { "zero": { } }
    }
  }
}
```

Instance

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "schema for BTL syntax",
    "type": "object",
    "oneOf": [
      {
        "properties": { "true" : { "additionalProperties": false } },
        "additionalProperties": false
      },
      ...
    ]
}
```

Schema

Object-program representation in **XML**

```
<pred xmlns="http://www.softlang.org/BTL">
  <if>
    <iszero><zero/></iszero>
    <succ><succ><zero/></succ></succ>
    <zero/>
  </if>
</pred>
```

Instance

```
<schema ...>
  <element name="true"><complexType/></element>
  <element name="false"><complexType/></element>
  ...
  <element name="if">
    <complexType>
      <group ref="tns:expr" minOccurs="3" maxOccurs="3"/>
    </complexType>
  </element>
  <complexType name="expr">
    <group ref="tns:expr"/>
  </complexType>
  <group name="expr"><choice>...</choice></group>
</schema>
```

Schema

The **metameta**level: the signature of signatures

```
type signature = decl* ;
symbol type : sort × typeexpr → decl ;
symbol symbol : fsym × typeexpr* × sort → decl ;
symbol boolean : → typeexpr ;
symbol integer : → typeexpr ;
symbol float : → typeexpr ;
symbol string : → typeexpr ;
symbol term : → typeexpr ;
symbol sort : sort → typeexpr ;
symbol star : typeexpr → typeexpr ;
symbol plus : typeexpr → typeexpr ;
symbol option : typeexpr → typeexpr ;
symbol tuple : typeexpr* → typeexpr ;
type sort = string ;
type fsym = string ;
```

There is one symbol for each construct of the extended signature notation described in ditto notation.

We use extended notation:

- *'*' for lists*
- *'**type**' for products / structural types*

N.B.: We may also be interested in the grammar of signatures and the signature of grammars and in a metamodel (i.e., a graph-based abstract syntax definition) of signatures etc.

Online resources

YAS (Yet Another SLR (Software Language Repository))

http://www.softlang.org/yas

YAS' GitHub repository contains all code.

See language BTL (for examples).

See languages BSL and ESL (for signature notations).

The Software Languages Book

http://www.softlang.org/book

The book discusses tree-based abstract syntax and object-program representation in detail.

Other related subjects:

concrete syntax, graph-based abstract syntax, …