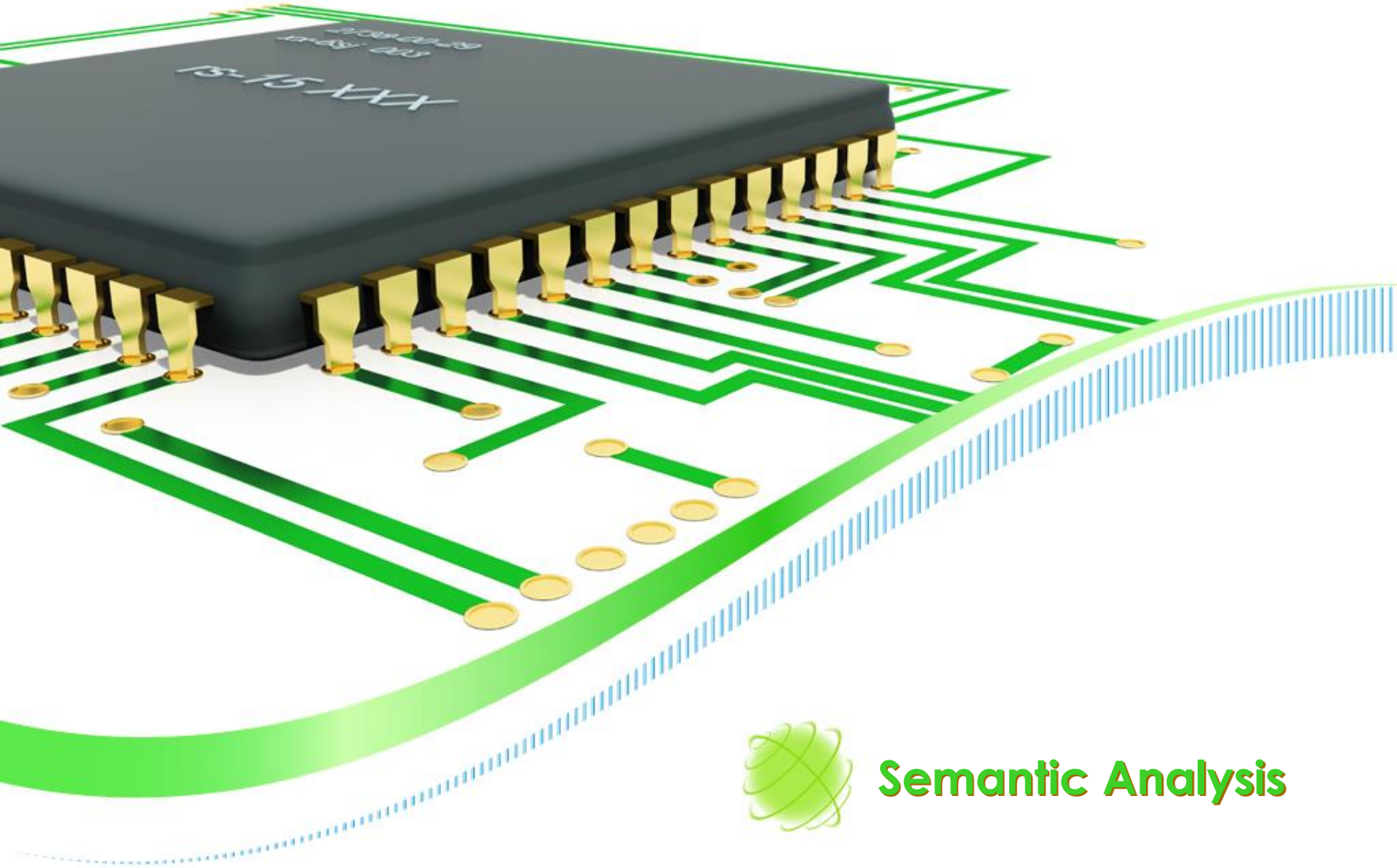
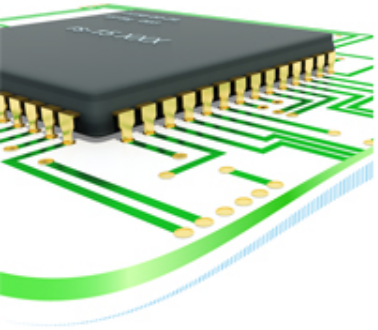


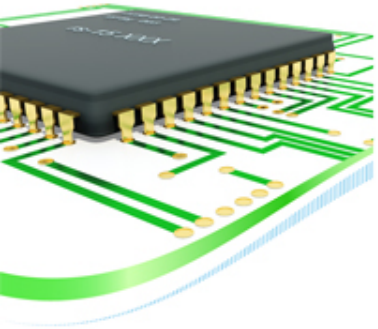
Semantic Analysis

Definition



Semantic analysis is the phase of checking context sensitive constraints and building the structures that represent the meaning of input.

It is also called as context sensitive analysis.



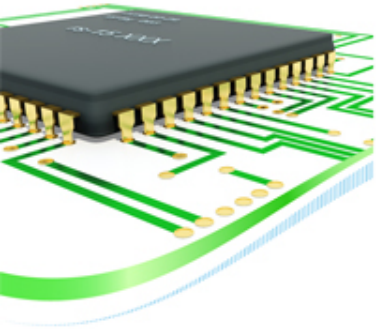
Semantic Analysis

Validity Revisited

Yes, valid program is a sentence.

Usually, validity transcends syntax.

- **Declarative Integrity**
- **Parameter Matching**
- **Type Conformance**
- **Operator Applicability**
- **Statement Applicability**
- **…**



Attribute Grammar

Parse Trees with Semantics

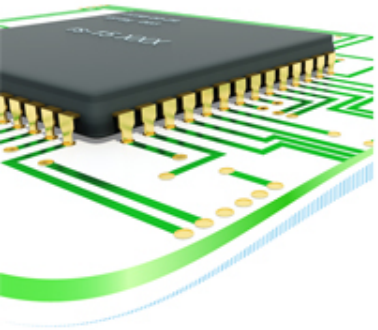
Nodes can have semantic annotations, which are known as attributes.

Syntax Directed Processing

- **Inherited Attributes (Top-Down)**
- **Synthesized Attributes (Bottom-Up)**

- **L-Attributed Grammar**
- **S-Attributed Grammar**

- **Opportune Moments in Parsing!**



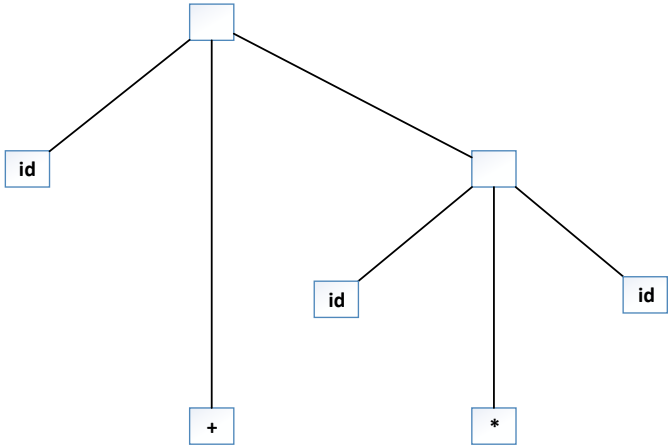
Abstract Syntax Tree (AST)

Bridging Syntax to Semantic Representation

Parse tree reflects derivations.

AST reflects logical structure.

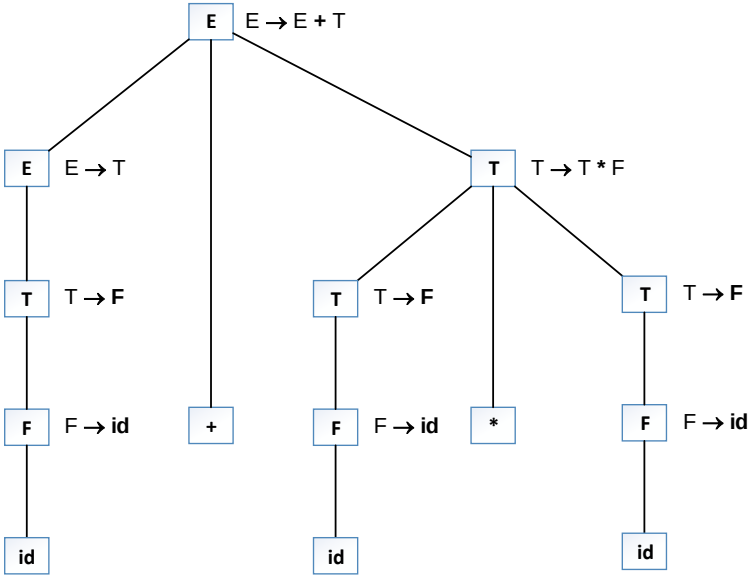
Input: id + id * id

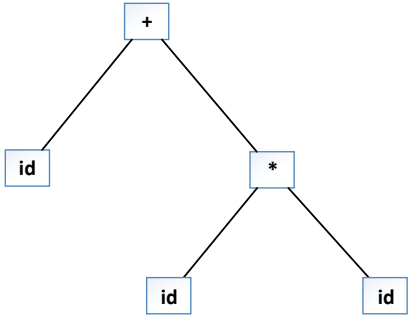


G:

E → E + T I T
T → T * F | F
F → - E I (E) I **id**

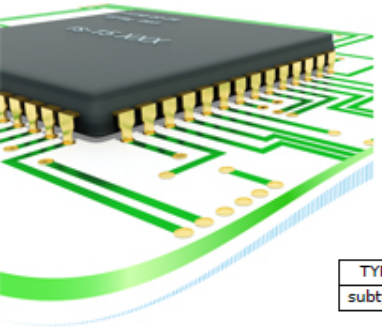
E → E + T
E → T
T → T * F
T → F
F → - E
F → (E)
F → **id**

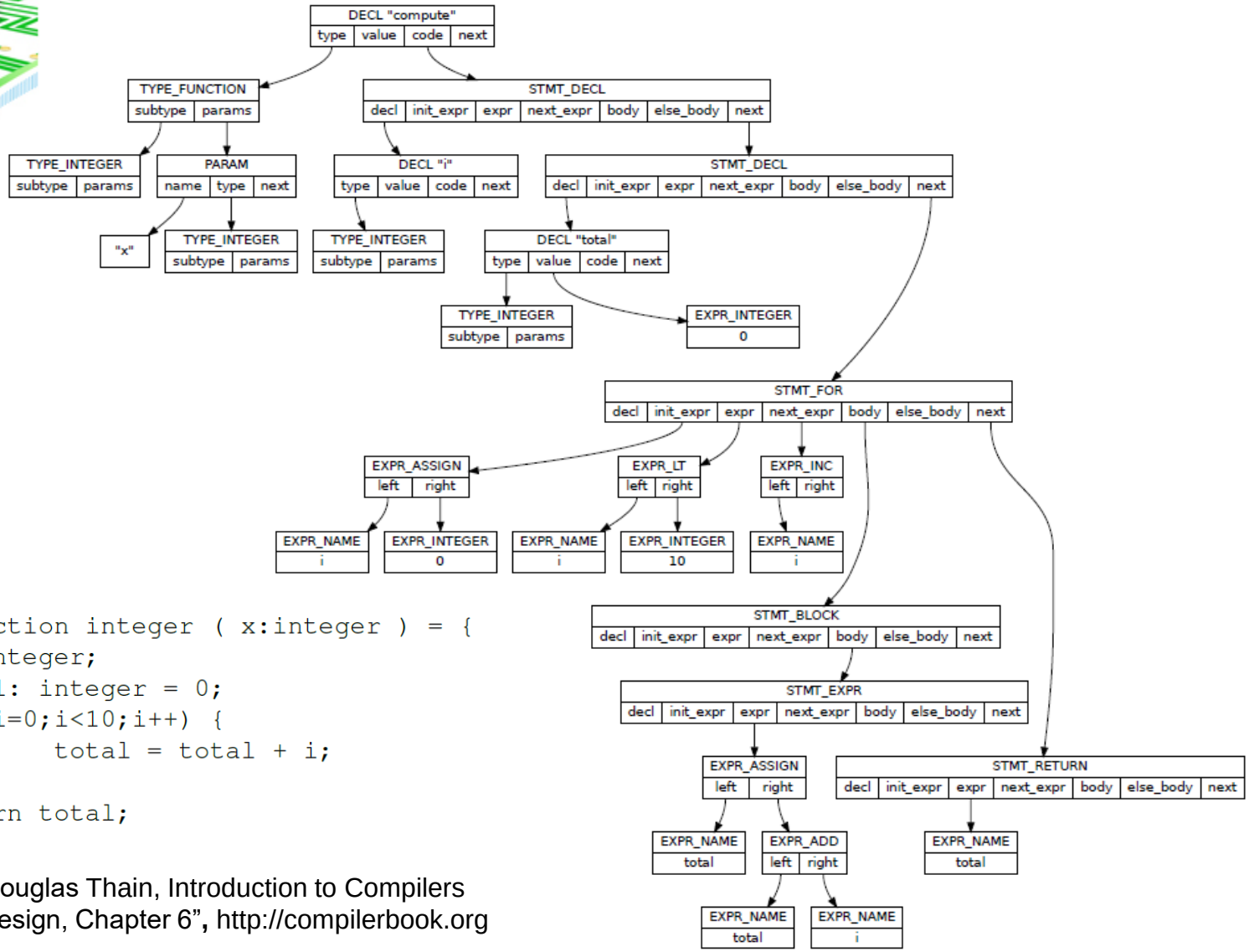




Abstract Syntax Tree (AST)

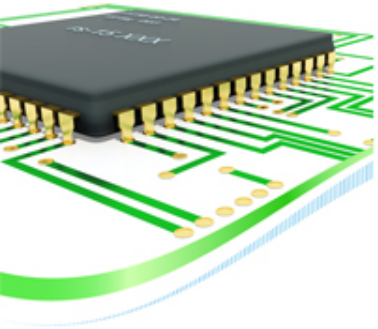
Bridging Syntax to Semantic Representation






```
compute: function integer ( x:integer ) = {
        i: integer;
        total: integer = 0;
        for(i=0;i<10;i++) {
                total = total + i;
        }
        return total;
}
```

Example from “Douglas Thain, Introduction to Compilers and Language Design, Chapter 6”, http://compilerbook.org

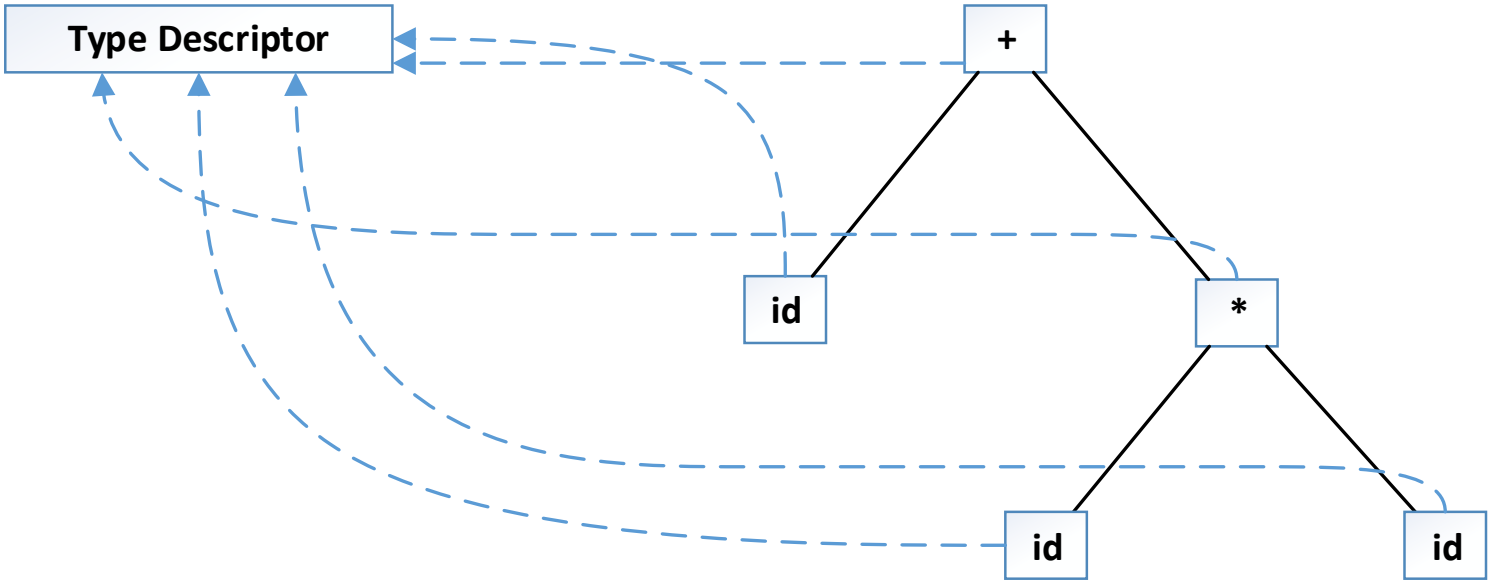


Abstract Syntax Tree (AST)

Bridging Syntax to Semantic Representation

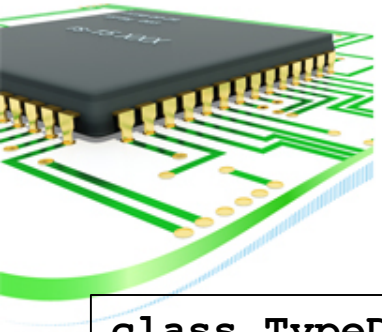
Syntax is hierarchical. **Meaning is not!**

- **Expressions**
- **Structure – Member Relationships**
- **Flow Control Statements**
- **Other Declarations**
- **…**



Representation

An Example



Meaning is complex!

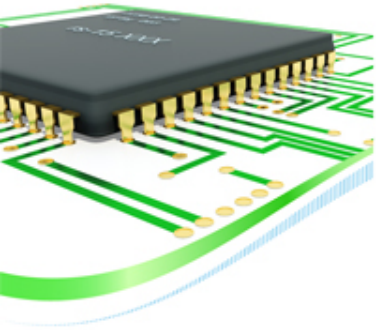
```
class TypeDescriptor
{
   TypeDescriptor    *baseType;
   int               dimension;
};
class ExpNode
{
   int       op;
   ExpNode   *left,
             *right;

};
class Statement
{
   int instructionOffset;
};
```

```
class EvaluationStatement : …
{
   ExpNode *exp;
};
class ForStatement : …
{
   ExpNode      *initExp,
                *conditionExp,
                *stepExp;
   Statement    *body;
};

class CompoundStatement : …
{
   list<Statement *> statList;
};
```

What to Represent



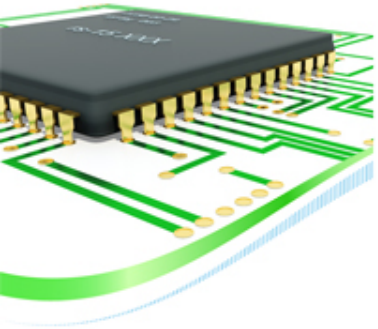
Depends on the problem that LP solves.
Complete with respect to input.
Compliant to the language specification.
Good enough to support subsequent phases.

For an Imperative Language

- **Types**
- **Statements**
- **Expressions**
- **Variables, Parameters**
- **Constants**
- **…**

For a Document Specification

- **Paragraphs**
- **Styles**
- **Objects**
- **Page Definition Data**
- **Constants**
- **…**



Type Systems

Defining Elements of Computation

A type system is complete and consistent set of rules that define the types available for computation.

Computation operates on values.

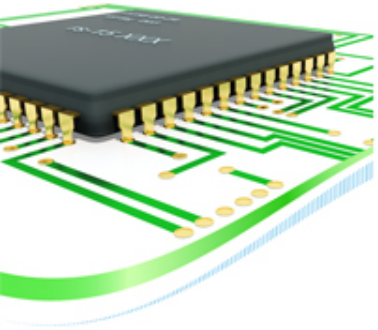
Values have certain properties like domain, limitations.

Properties are defined by types.

Types are contained by the type systems.

Type Systems

Defining Elements of Computation



Any type is eventually expressed in terms of data types supported by underlying computing architecture.

Architectural layers as hardware and software.

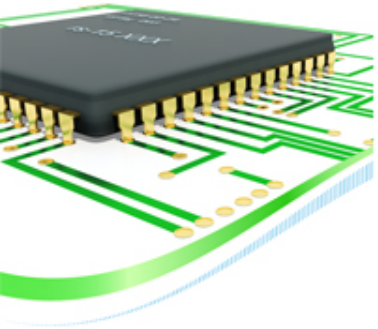
The LPs capability of utilizing architectural layers!

Example cases:

The 8086 – 8087 co-processor coupling.

GPUs, DPUs, TPUs, …

Language-native architecture-alien types (DBMS, R, JS, ….)



Type Systems

Defining Elements of Computation

What to know about a type system

Type Manipulation

What are the basic types?

Is type derivation possible? If yes how?

Rules for type equivalence?

What are the implicit / explicit type conversion rules.

Is type inference applicable?

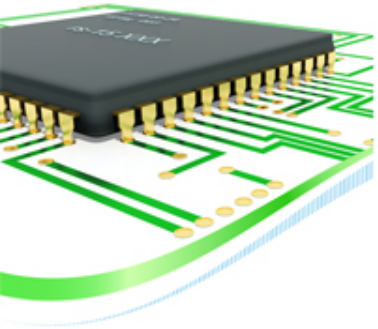
Basic distinctions and challenges

Static / Dynamic Type Checking

Strong / Weak Type Checking

Safe Programming

Extensible / Fixed



Type Systems

Defining Elements of Computation

How to implement

Type system specification is connected to

Symbol Table Processing

Type Dependent Semantic Checks

Type Arithmetic

Code Generation

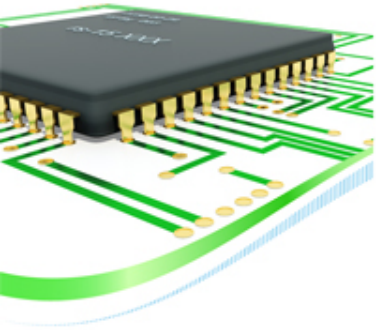
Language Runtime

Abstract Architectural Extensions

Compile Time vs Runtime

Symbol Management

Items to extract semantics from



Symbol table is the central repository creating the capacity to store and restore data about a given symbol and scope.

Semantic analysis builds a comprehensive data structure to store entities like types, classes, structures, variables, parameters, and more.

- **Efficient Management**
- **Efficient Name Resolution**
- **Name Spaces, Scopes**