Making Your Own Domain Specific Language

A **domain-specific language** (**DSL**) is a computer language specialized to a particular application **domain**. This is in contrast to a ***general-purpose language*** (GPL), which is broadly applicable across domains, and lacks specialized features for a particular domain.

--Wikipedia

Domain: A sphere of knowledge (ontology), influence, or activity. The subject area to which the user applies a program is the domain of the software.

--Wikipedia

Mitchell Harris

@heneryville

github.com/heneryville/MineDefine

Domain Specific Languages You've Heard Of

- Http
- Url
- SQL
- XML
- HTML
- JSON
- CSS/selectors
- Numerals
  - Hex
  - Decimal, commas
  - Roman numerals
  - Scientific notation
- Dates
- Addresses
- CSV
- Pig/Hive
- Mathematic Notation (e.g. Integrals)
- Phone Numbers
- LINQ
- Web Routes
- Bar Codes
- QR codes
- Markdown
- Twitter hash tags and @
- UML
- YAML
- Version numbers (2.0.1)
- IP address
- Excel equations
- Regular Expressions
- Ant/NANT
- xPath, xQuery
- jPath
- XSLT
- Graphics turtle language
- awk & sed
- Make
- Post script
- PHP ini
- Nginx
- Scripture book,chapter,verse numerology

CSS (declarative DSL)

```css
div.container .item:nth-child(even) {
    background-color: red;
}
```

JS (General Purpose)

```javascript
var divElems = document.getElementsByTagName("div");
for (var i=0; i<divElems.length; ++i) {
    if(divElems[i].className.split(/\s+/).indexOf('container') < 0) continue;
    var children = divElems[i].children;
    for (var j=0; j<children.length; ++j) {
        if(children[j].className.split(/\s+/).indexOf('item') < 0) continue;
        if(j%2 == 0) continue;
        children[j].style.backgroundColor = "blue";
    }
}
```

An Internal DSL

```
private InlineContainer testData = new InlineContainer()
            .Person(new {id ="1", dob = "1991", gender = "male"}).Up
            .Person(new {id ="2", dob = "1992", gender = "female"}).Up
            .Person(new {id ="3", dob = "1993", gender = "male"}).Up
            .Person(new {id ="4", dob = "1994", gender = "female"}).Up
            .Family("1","2","3").Up
            .Family("3","4").Up
            ;
```

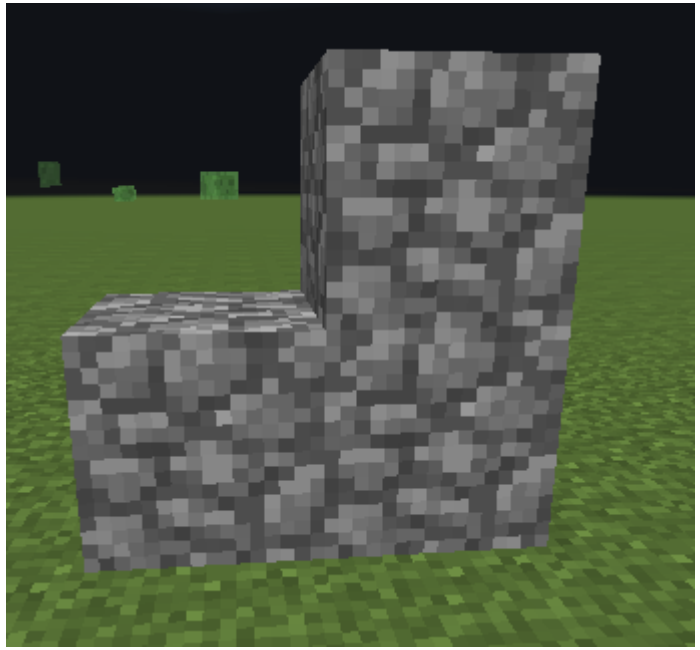
MineDefine

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```

github.com/heneryville/MineDefine

MineDefine

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```



github.com/heneryville/MineDefine

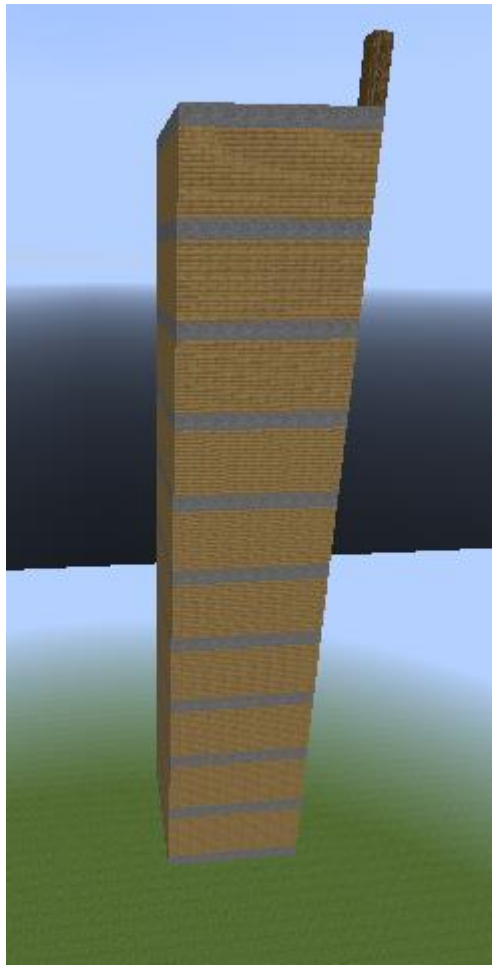
MineDefine

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```

github.com/heneryville/MineDefine

MineDefine

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```



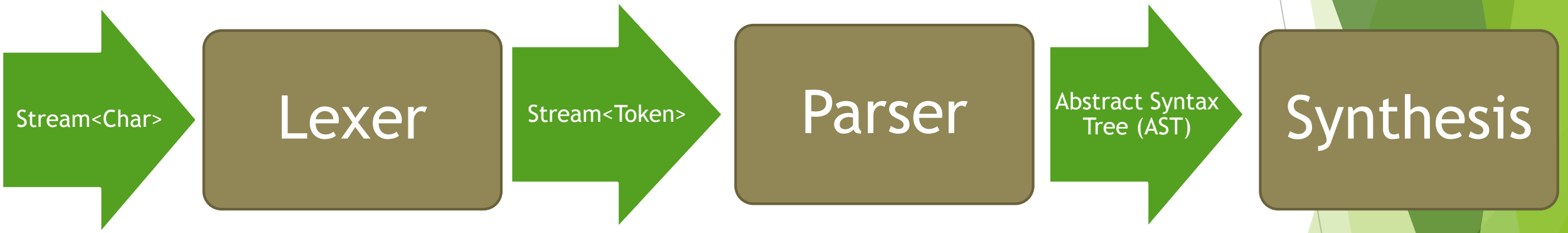
github.com/heneryville/MineDefine

MineDefine

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```

github.com/heneryville/MineDefine

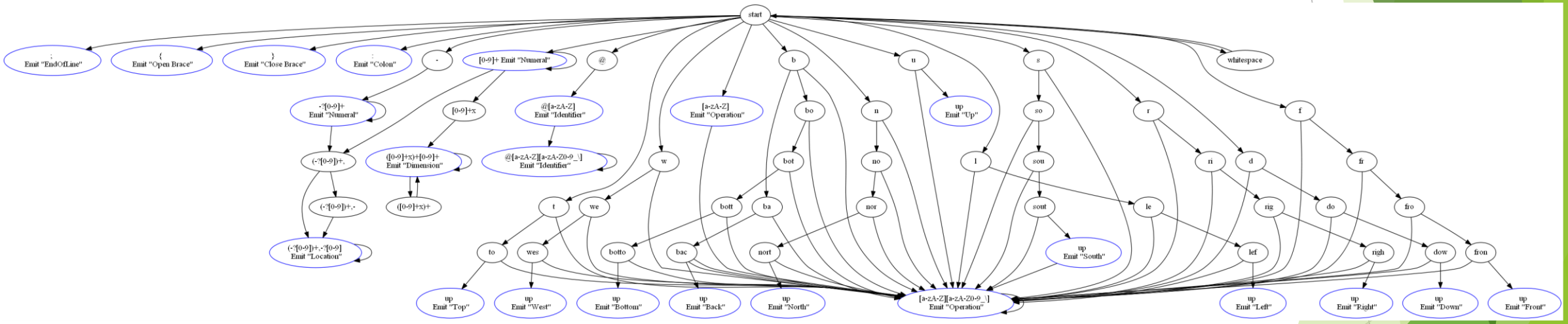
Parts of a Compiler



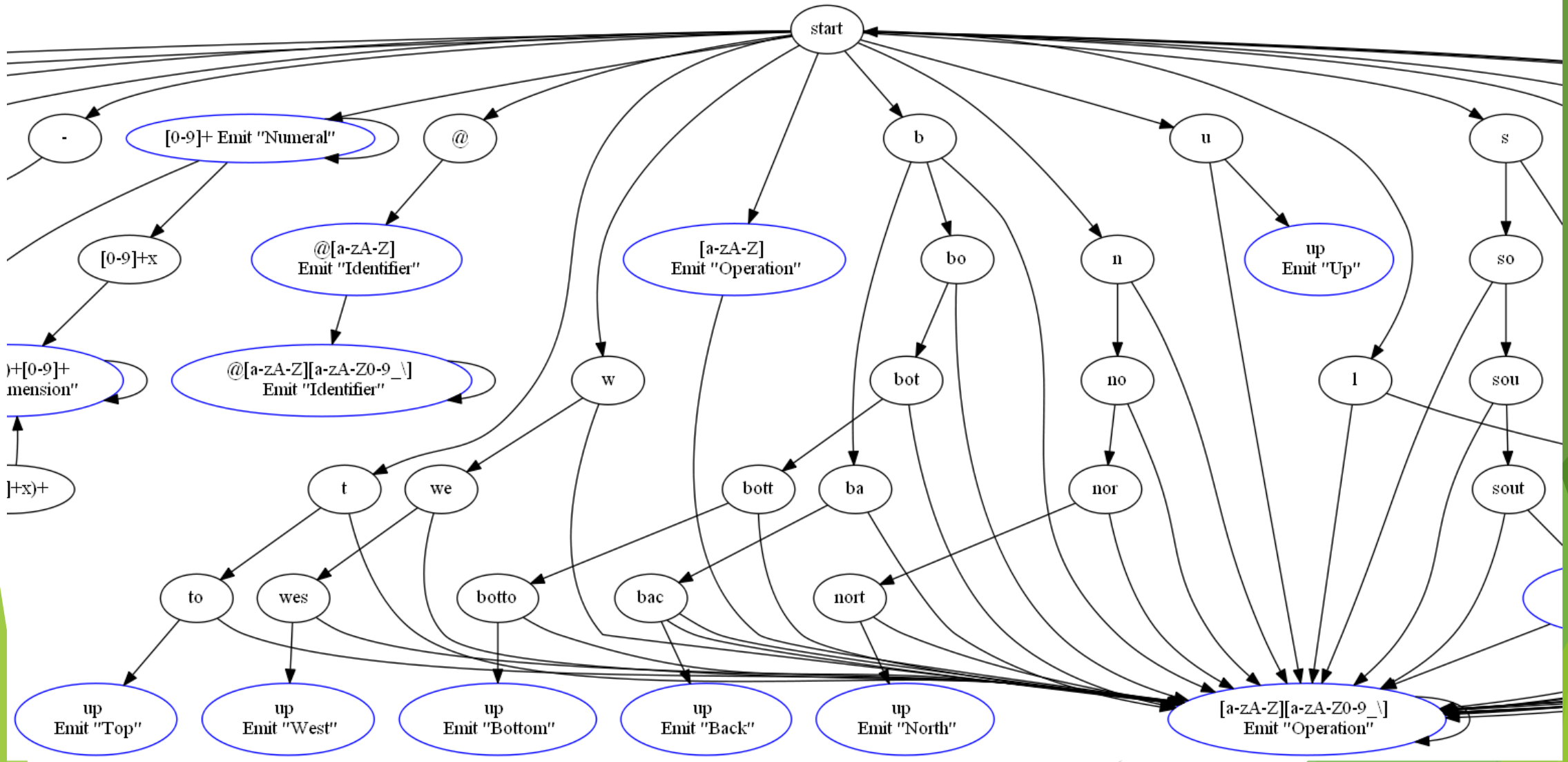
Super Simple DSL

```
int Parse(string text)
{
    return text.Split(';')  //Lex
        .Select (x => int.Parse(x)) //Parse
        .Sum();  //Synthesis
}
```

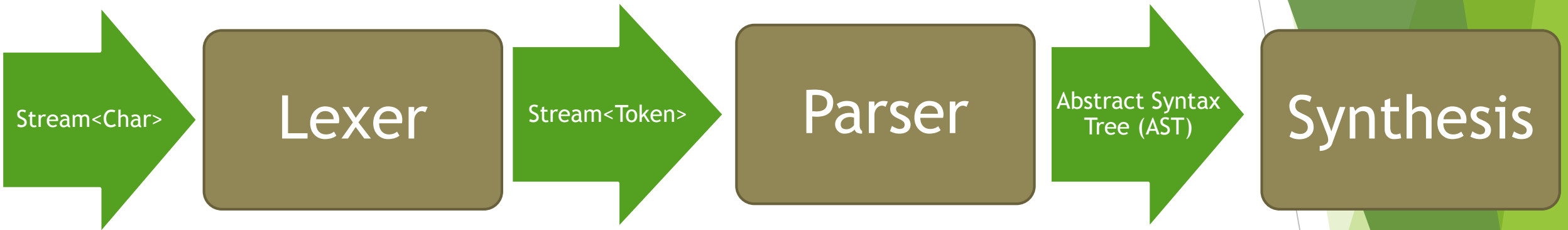
MineDefine Finite State Machine



MineDefine Finite State Machine



Parts of a Compiler



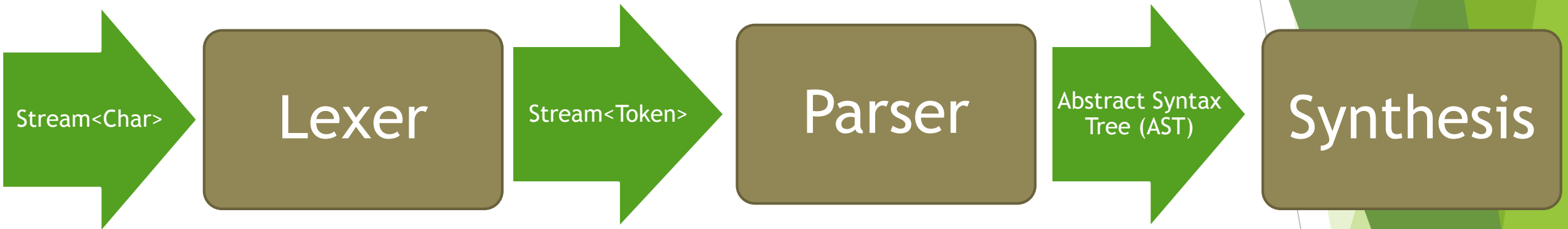

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```

```
Identifier (@building), Colon,
OpenBrace, Identifier (@chair), Colon,
OpenBrace, Dimension (2x1), Identifier
(@cobblestone), EndOfLine, Up,
Identifier (@cobblestone), EndOfLine,
CloseBrace, Identifier (@floor), Colon,
OpenBrace, Dimension (10x10x1),
Identifier (@stone), EndOfLine, Up,
EndOfLine, Location (4,5), Identifier
(@chair), EndOfLine,
Operation (wall), Dimension (10x10x4),
Identifier (@woodplank), EndOfLine,
CloseBrace, Identifier (@roof), Colon,
Dimension (10x10x1), Identifier
(@stone), EndOfLine, Dimension (1x1x10),
Identifier (@floor), EndOfLine, Top,
Identifier (@roof), EndOfLine, Top,
Dimension (1x1x3), Identifier (@wood),
EndOfLine, CloseBrace, Identifier
(@building), EndOfLine
```

MineDefine BNF

```
<prgm> ::= <statement>*
<statement> ::= <definition> | <invocation>| <transform>
<definition> ::= identifier: <statement_block>
<statement_block> ::= <statement> | { <statement>* }
<transform> ::= <trans_instructions> <statement_block>|<trans_instructions>;

<trans_instructions> ::= <absoulte_trans> | <relative_trans>
<absolute_trans> ::= (up|down|north|south|east|west) integer?
<relative_trans> ::= (top|bottom|left|right|front|back)

<invocation> ::= <shape>? <dimension>? <location>? identifier;
```

Parts of a Compiler



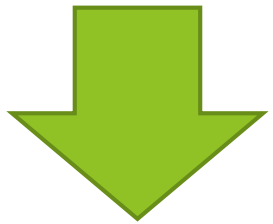

```
@building: {

  @chair: {
    2x1 @cobblestone;
    up @cobblestone;
  }

  @floor: {
    10x10x1 @stone;
    up;
    4,5 @chair;
    wall 10x10x4 @woodplank;
  }
  @roof: 10x10x1 @stone;

  1x1x10 @floor;
  top @roof;
  top 1x1x3 @wood;
}

@building;
```

```
Identifier (@building), Colon,
OpenBrace, Identifier (@chair), Colon,
OpenBrace, Dimension (2x1), Identifier
(@cobblestone), EndOfLine, Up,
Identifier (@cobblestone), EndOfLine,
CloseBrace, Identifier (@floor), Colon,
OpenBrace, Dimension (10x10x1),
Identifier (@stone), EndOfLine, Up,
EndOfLine, Location (4,5), Identifier
(@chair), EndOfLine,
Operation (wall), Dimension (10x10x4),
Identifier (@woodplank), EndOfLine,
CloseBrace, Identifier (@roof), Colon,
Dimension (10x10x1), Identifier
(@stone), EndOfLine, Dimension (1x1x10),
Identifier (@floor), EndOfLine, Top,
Identifier (@roof), EndOfLine, Top,
Dimension (1x1x3), Identifier (@wood),
EndOfLine, CloseBrace, Identifier
(@building), EndOfLine
```

```
Program
.Definition: building
..Definition: chair
...Build: 2x1x1 cobblestone
...Transform: Up 1
....Build: cobblestone
..Definition: floor
...Build: 10x10x1 stone
...Transform: Up 1
...Build: 4,5,0 chair
...Build: Wall 10x10x4 woodplank
..Definition: roof
...Build: 10x10x1 stone
..Build: 1x1x10 floor
..Transform: Top
...Build: roof
..Transform: Top
...Build: 1x1x3 wood
.Build: building
```

Lowering (Syntactic Sugar)

```
for(int i=0; i<10; ++i) {
    //Do Something
}
```



```
{
    int i=0;
    while(i<10) {
        //Do something
        ++i;
    }
}
```