Weave: A Dynamic Multi-Language Parsing Framework

Mitchel Pyl

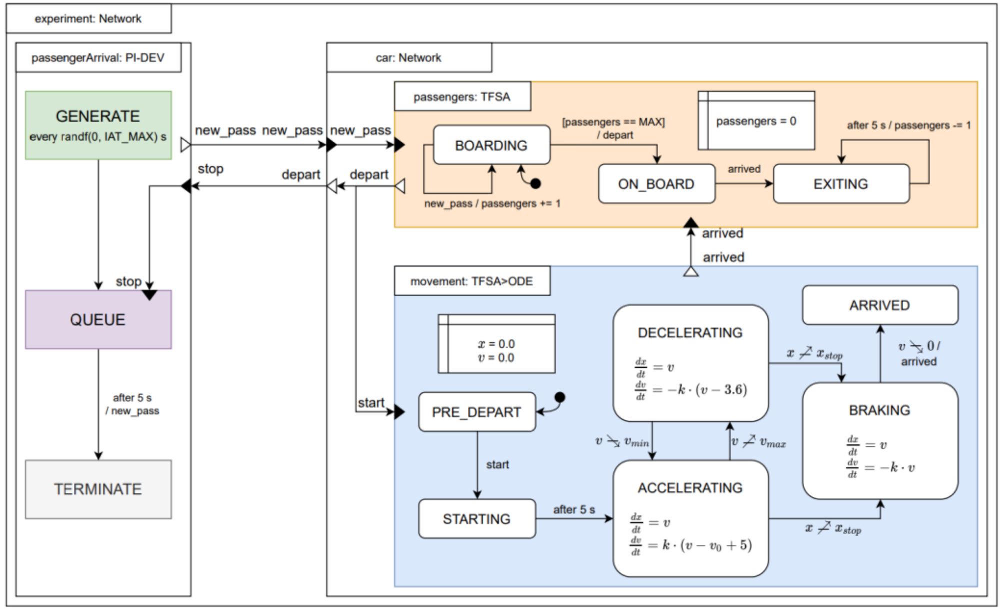
Promotor: Hans Vangheluwe

Introduction

- **Model**
  - Abstraction of relevant properties of a (real-life) system
- **Formalism**
  - Collection of abstract syntax, concrete syntaxes, semantics
- **Multi-Paradigm Modelling**
  - Modelling using the most appropriate formalism(s) at the most appropriate levels of abstraction, explicitly modelling workflows
- **Hybrid languages**
- **Hybrid models**

Hybrid Languages

- Combine abstract syntax, semantics, and concrete syntax!
- Previous work already focused on AS, semantics, and visual CS
- We looked at textual CS



[1] Paredis et al.

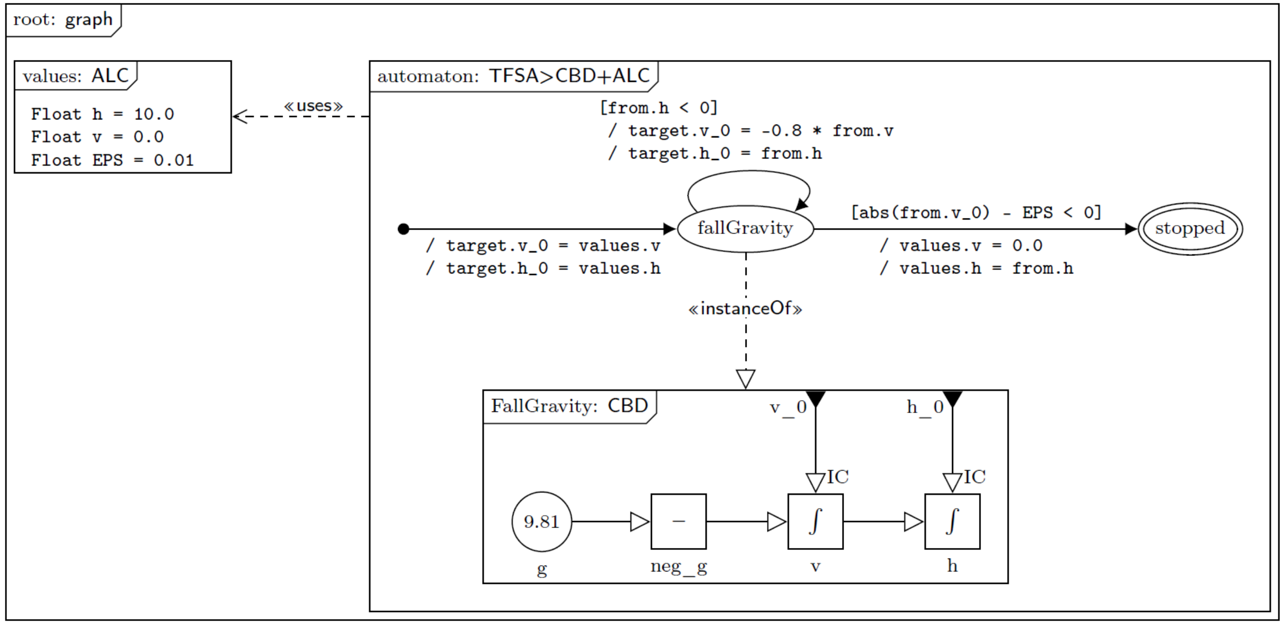
```
values = >alc#block{
    Float h = 10.0
    Float v = 0.0
    Float EPS = 0.01
}

automaton = >tfsa:pp{
    FallGravity = >cbd{
        parameter h_0;
        parameter v_0;
        constant g = 9.81;

...
```

Goals

- **Specify textual syntax for hybrid languages**
- **Parse hybrid languages**
  - Multi-language parsing
- **Allow dynamically selecting which language**
- **Allow dynamically defining new languages, and then using them**
- **Integrate with the Modelverse**

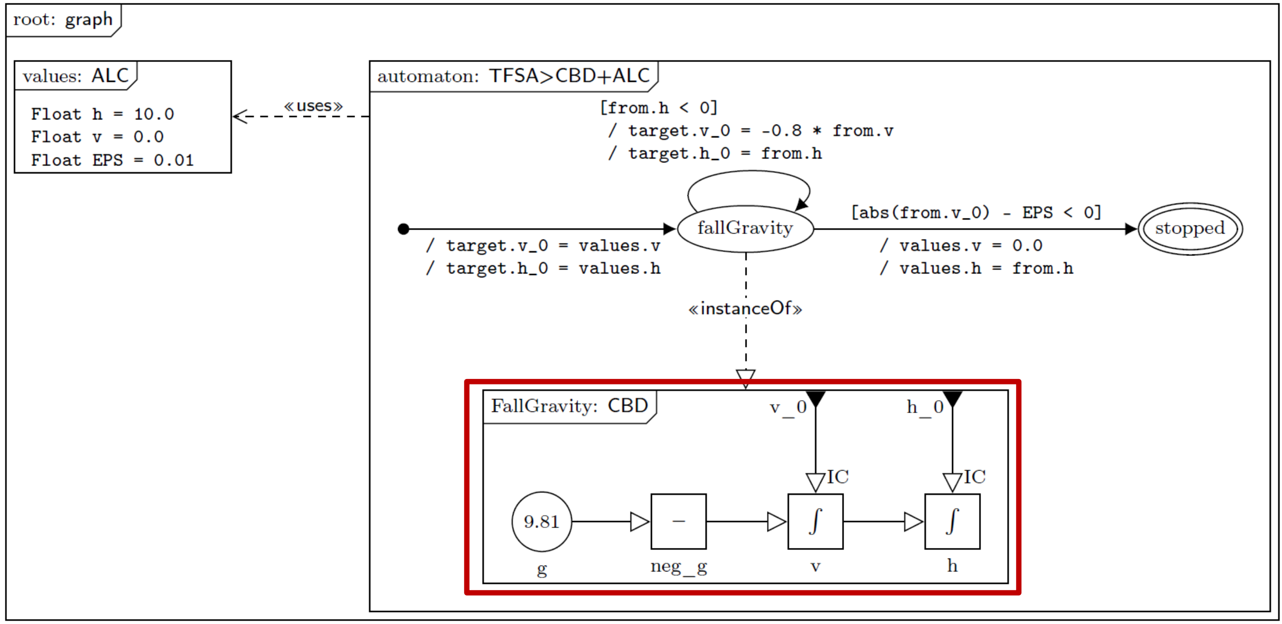
Example Model

- **Hybrid TFSA-CBD Model**
  - Bouncing ball example



Example Model

```
parameter h_0;
parameter v_0;
constant g = 9.81;

block v: Integrator;
block h: Integrator;
block neg_g: Negator;

connect g     to neg_g;

connect neg_g to v;
connect v_0   to v.IC;

connect v     to h;
connect h_0   to h.IC;
```

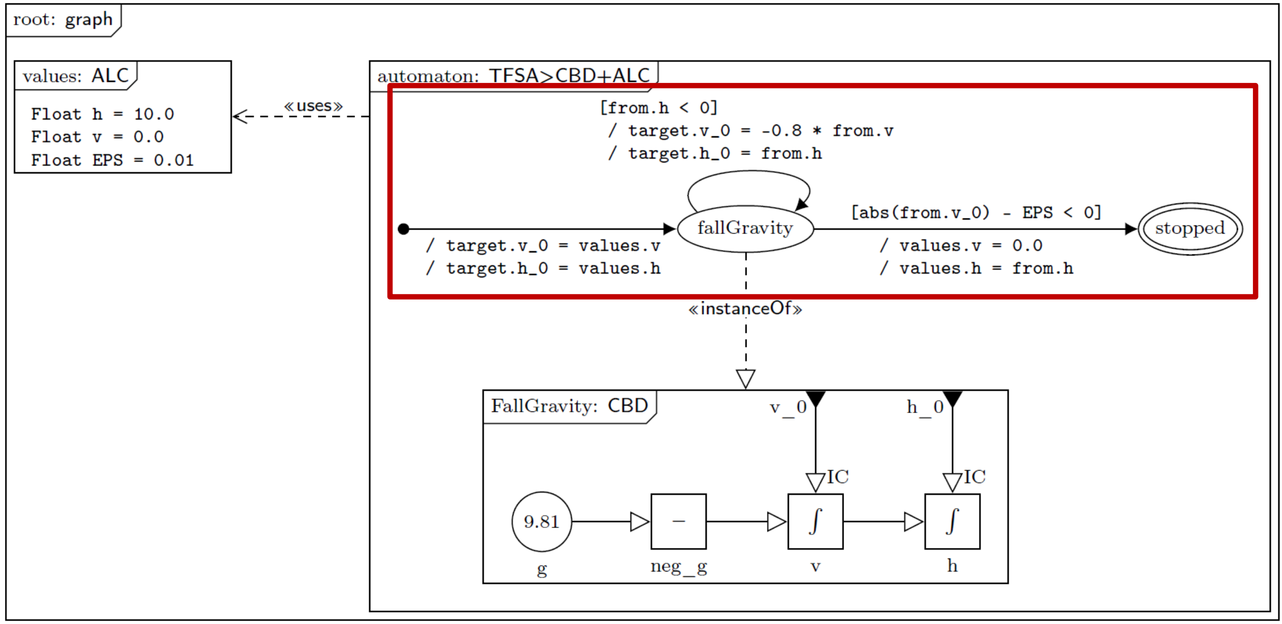


- **Causal Block Diagram (CBD)**
- **Simulates free-fall**

- $\frac{dv}{dt} = -g$ $\qquad \mathrm{v} = \mathrm{v}_0 + \int_0^t -g \, dt$
- $\frac{dh}{dt} = v$ $\qquad \mathrm{h} = \mathrm{h}_0 + \int_0^t v \, dt$

Example Model

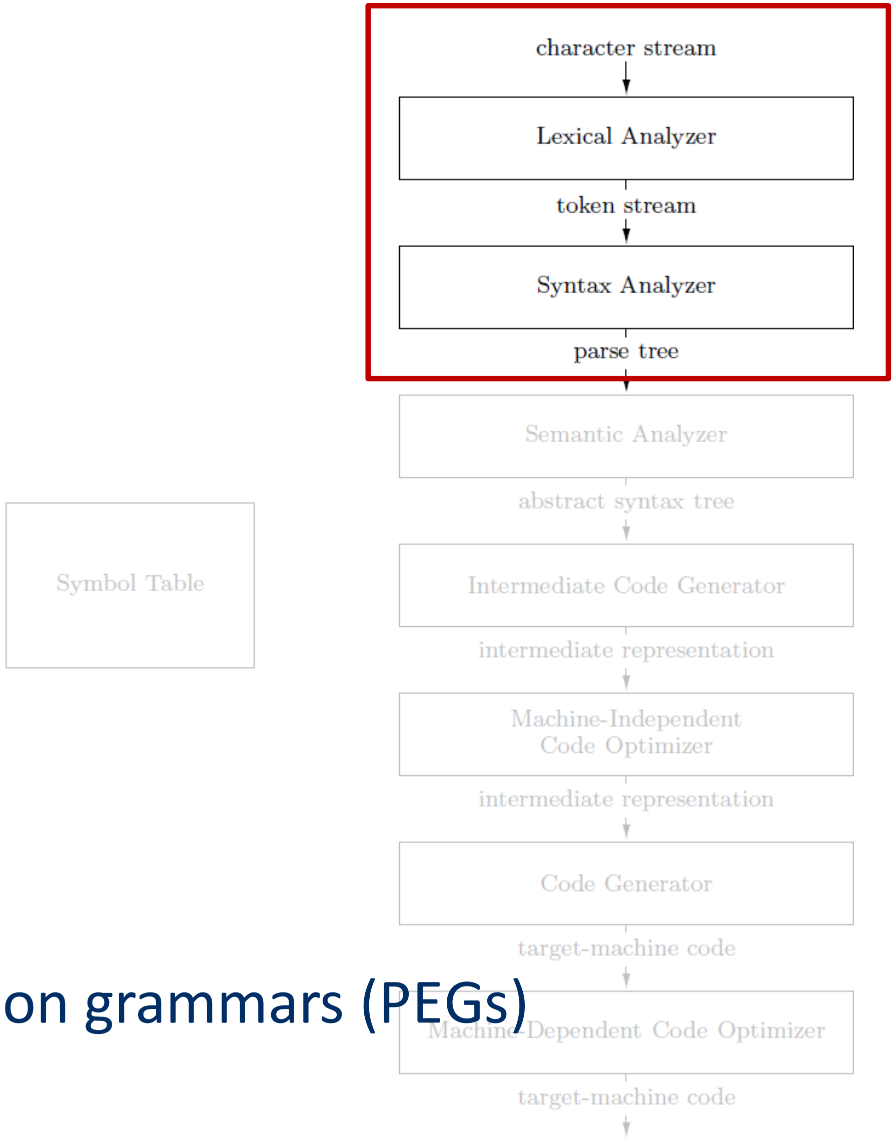
```
initial state start;
state fallGravity: FallGravity;
final state stopped;

transition from start to fallGravity
    do {
        target.v_0 = values.v
        target.h_0 = values.h
    };

transition from fallGravity to fallGravity
    when [fallGravity.h < 0]
    do {
        target.v_0 = -0.8 * from.v
        target.h_0 = from.h
    };

transition from fallGravity to stopped
    when [abs(fallGravity.v_0) - EPS < 0]
    do {
        values.v = 0
        values.h = from.h
    };
```



- **Timed Finite-State Automaton (TFSA)**
- **States**
- **Transitions**
- **Guards in Action Language**
- **Actions in Action Language**

General Parsing Architecture

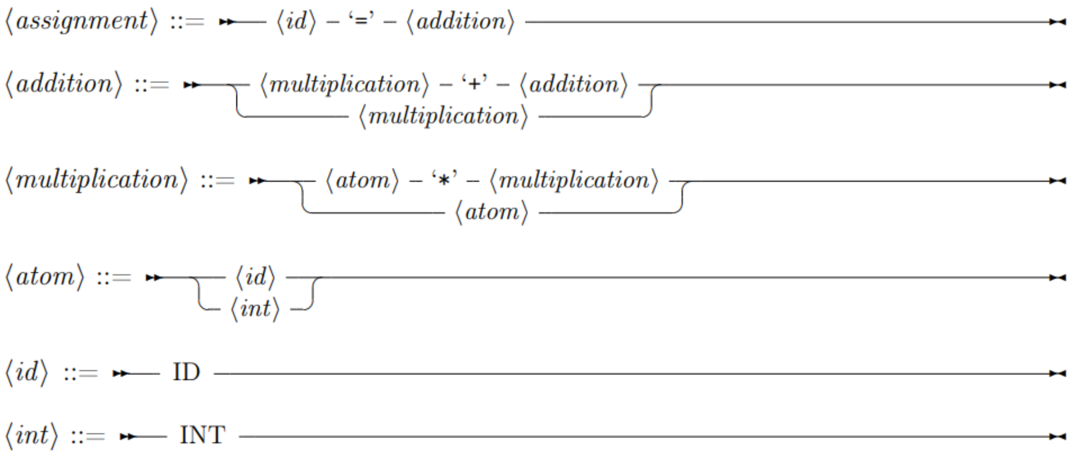
- **Classically using lexer and parser**
- **Lexer**
  - Takes: character stream
  - Gives: token stream
  - Uses: regular languages (e.g. regular expressions)
- **Parser**
  - Takes: token stream
  - Gives: parse tree
  - Uses: context-free grammars (CFGs), parsing expression grammars (PEGs)



[2] Aho et al.

Parse Trees

- Simple tree structure
- Leaf nodes labelled by a terminal or empty symbol
- Interior nodes labelled by a non-terminal
- Root node labelled by start symbol
- Some drawbacks



⟨assignment⟩ ::= ⟨id⟩ – ‘=’ – ⟨addition⟩

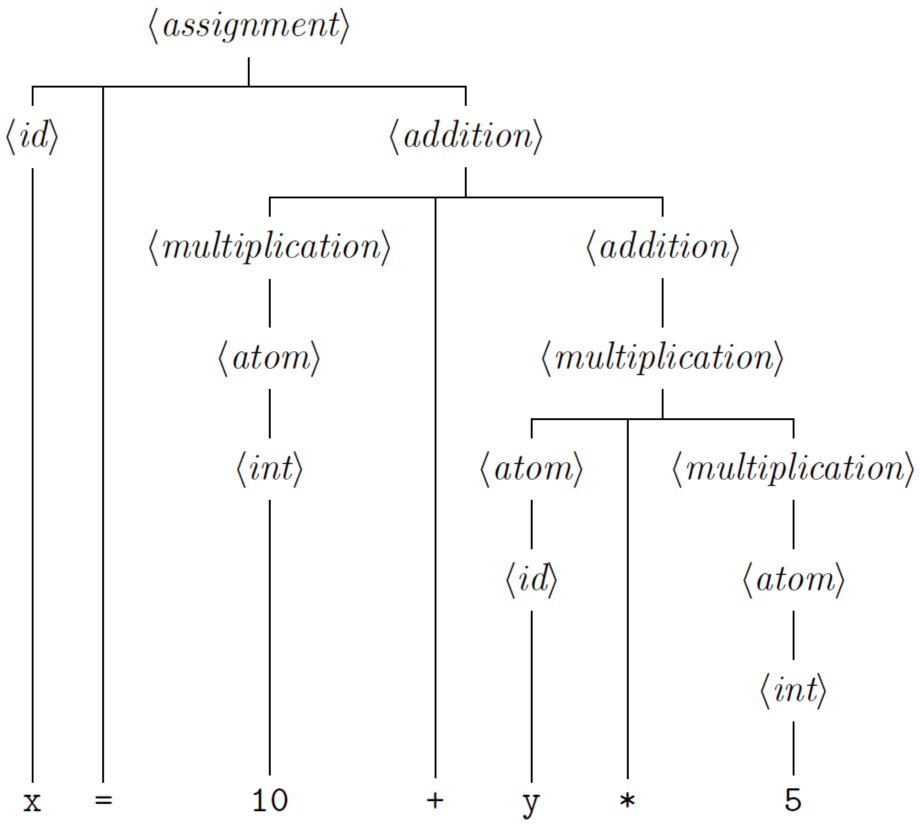
⟨addition⟩ ::= ⟨multiplication⟩ – ‘+’ – ⟨addition⟩ | ⟨multiplication⟩

⟨multiplication⟩ ::= ⟨atom⟩ – ‘*’ – ⟨multiplication⟩ | ⟨atom⟩

⟨atom⟩ ::= ⟨id⟩ | ⟨int⟩

⟨id⟩ ::= ID

⟨int⟩ ::= INT

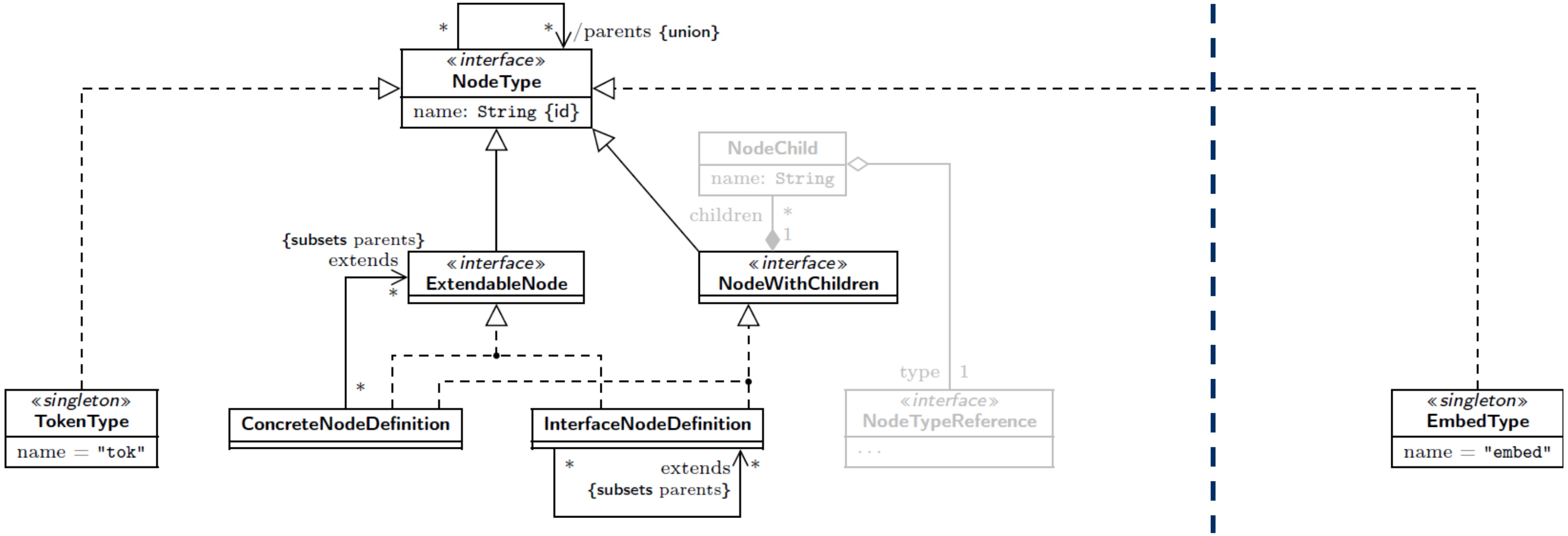


Parse Trees

- **Shape of interior node based on production rule**
  - Changes to the grammar always change the tree shape
- **eBNF: theoretically infinite possible shapes**
  - Kleene star and Kleene plus operations
- **Tree children ordered by occurrence**
- **Complex grammars result in complex trees**
- ***Does not account for hybrid languages!***

Tree Definition Formalism

- Type-based trees
- Decoupled from grammar
- Node children accessible through assigned name, not index or type
- Simple type hierarchy, basic polymorphism
- Data only, no operations



Base TDF

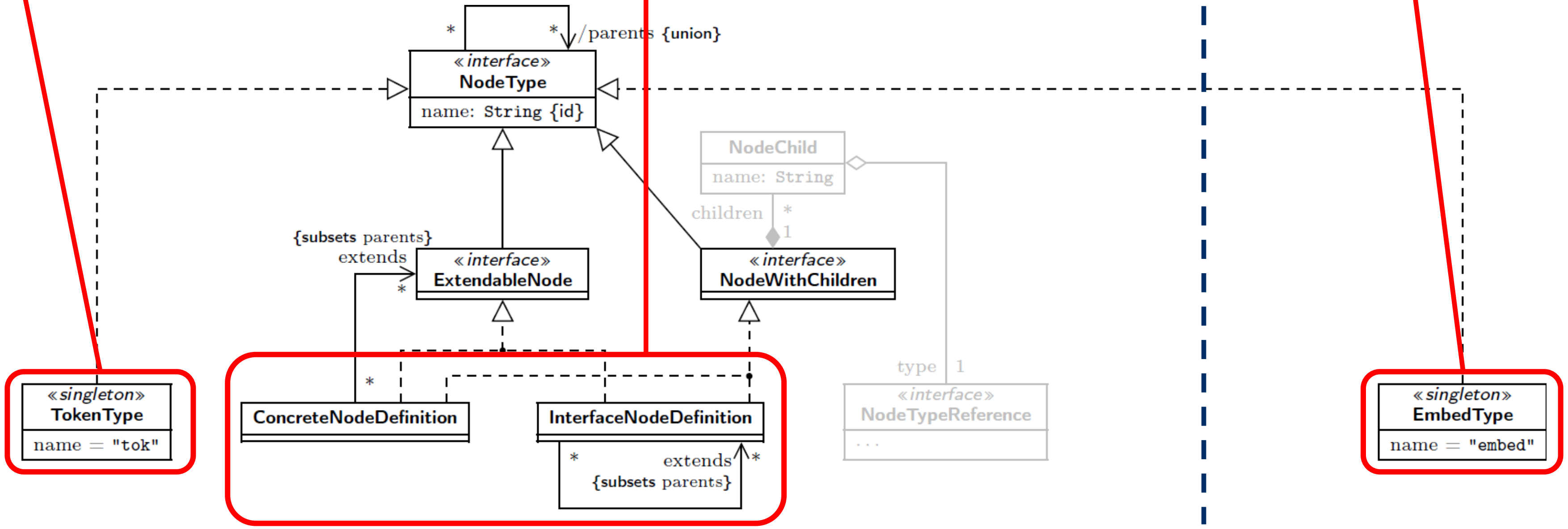
Embedding Extension

Tree Definition Formalism

Tokens, Lexer output

Interior nodes, Parser output

Embedded models, sub-parser output

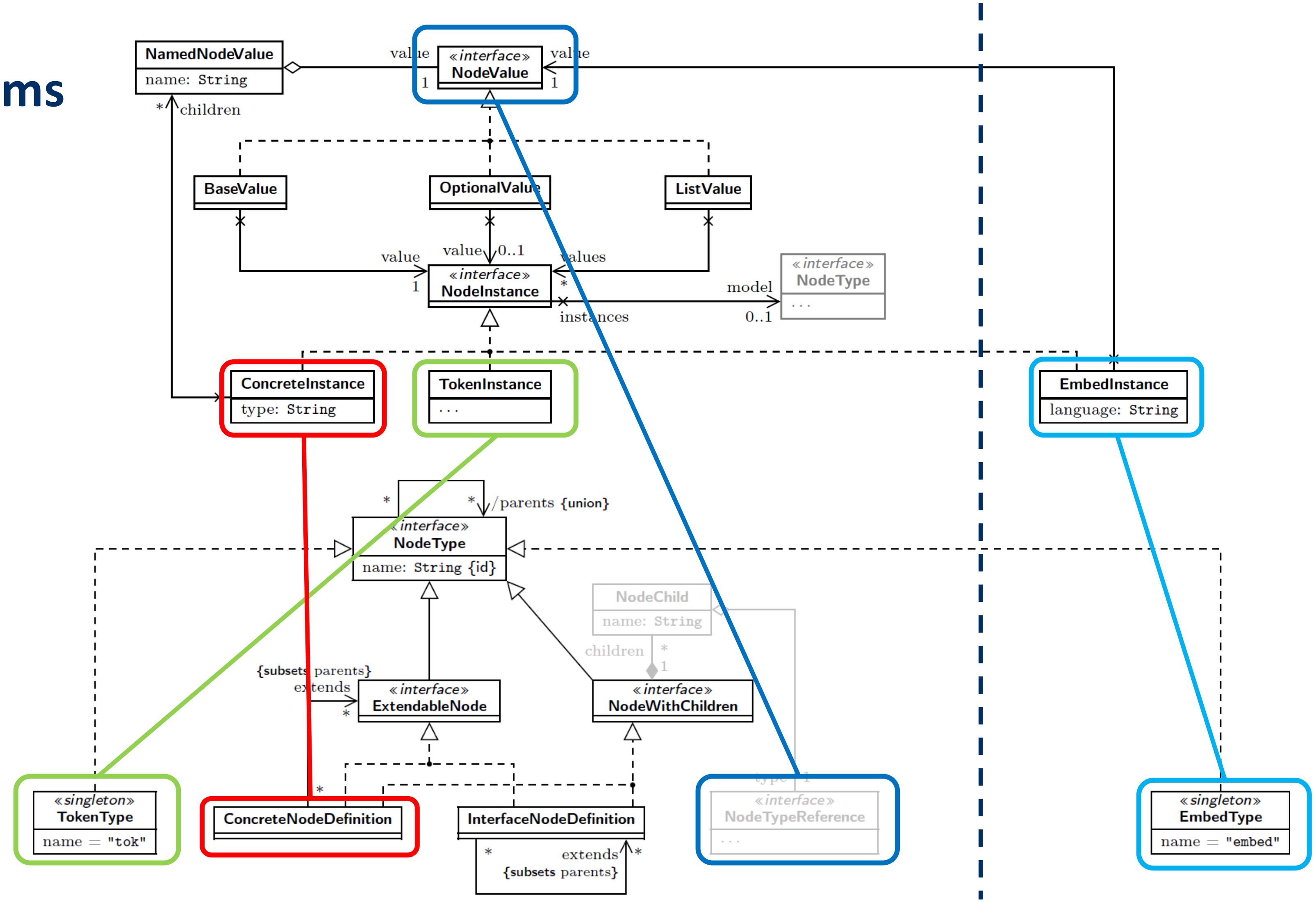


Base TDF

Embedding Extension

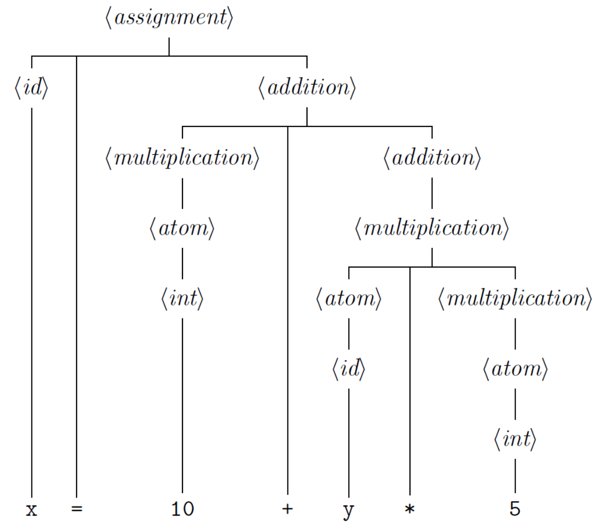
Tree Instance Formalism

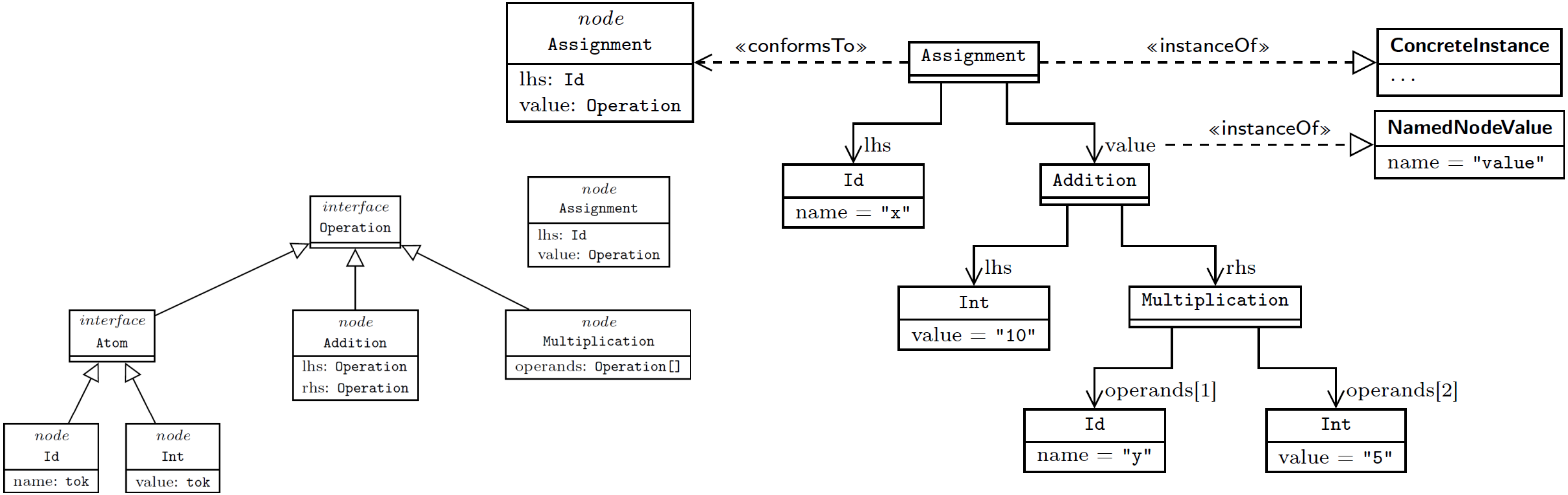
- **Is to TDF like object diagrams are to class diagrams**



Tree Instance Formalism

```
x = 10 + y * 5
```



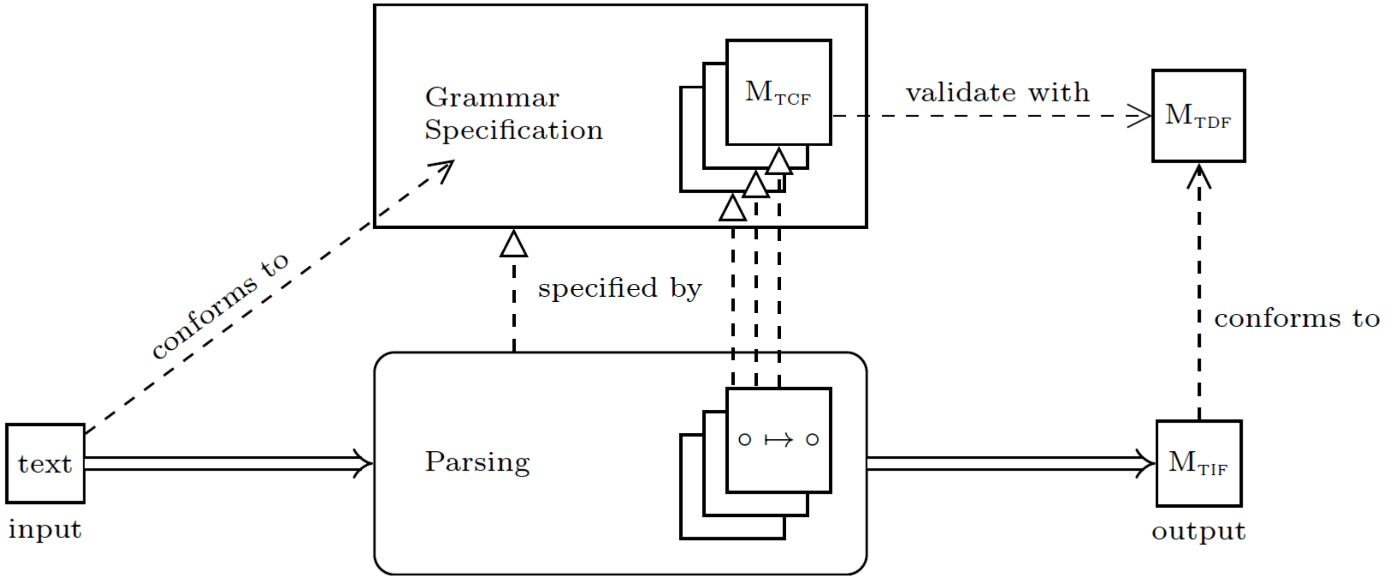


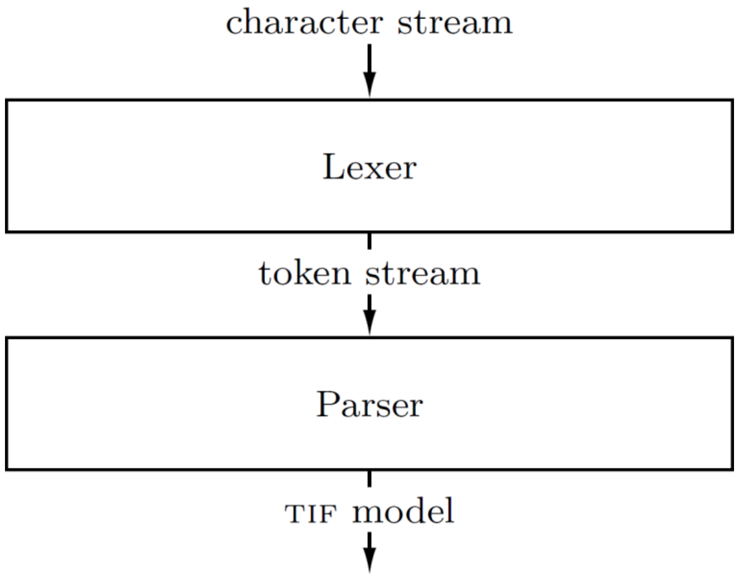
Tree Construction Formalism

- **Declarative method of building a tree**
- **Not a programming language**
  - Data flow, not control flow
- **Independent of the implementation language**

Parse Trees

- **TDF, TIF and TCF used as replacement for building parse trees**





Multi-Language Parsing

- **Different languages have different syntax, different constraints**
- **Cross-language variability**
  - Grammar class
  - Indentation sensitive (layout constraints, indent-dedent tokens) or not
  - Character set (ASCII, Unicode)
  - Keywords, operators, separators, delimiters
  - Comment styles
- **Difficult to combine languages**
- **Use sub-parsers instead**
- **But: need to be able to specify where “language fragments” can appear**

Multi-Language Parsing

- **Language fragments specification as part of the grammar**
- **Normal grammar:**
  - Terminals
  - Non-terminals
- **Add "embed specifications"**
  - Has its own behaviour
- **Lives on same level as terminals, as they describe "content"**

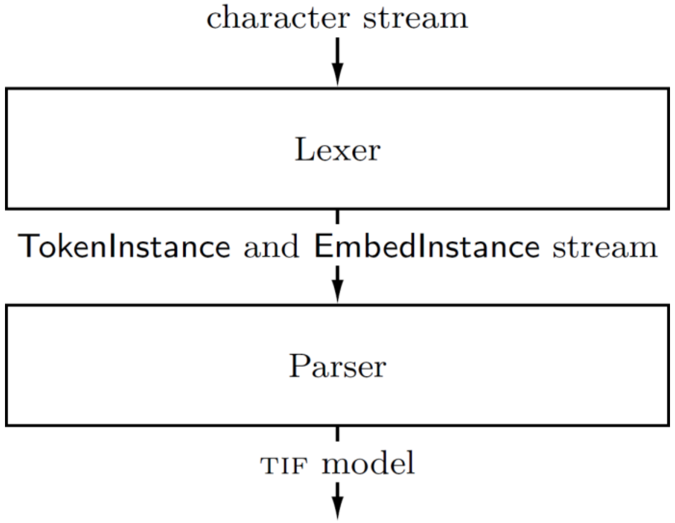
Multi-Language Parsing

- **When switching languages, lexer needs to relinquish control over the character stream**
- **Parser is best suited for deciding where to switch languages, to which language**
- **Parser can backtrack**
  - May cause lexer to backtrack as well
  - May attempt to parse fragment, or retry as not a fragment
- **Need communication from parser to lexer**

Multi-Language Parsing

- Workaround: make lexer decide instead
- Limit syntax to something the lexer can handle
- Statically select language (fixed as part of the host language)
  - "[" {*specific language*} "]"
- Dynamically select language
  - ">" {*language selection*} "{" {*fragment content*} "}"

- Token stream no longer contains tokens only



Implementation

- **Lark: Python parser generator library**
  - Supports on-the-fly parser creation
  - Allows manipulation of parse tree creation, lexing, token stream
- **WEAVE framework as a**
  - collection of languages
    - TDF and TCF for parse trees
    - Weave grammar language
  - API for language switching
  - parser generator for hybrid parsers
    - Using Lark, but supporting other backends as well

Example Weave Language: Simple TFSA

Weave Grammar Specification

```
start[TimedFiniteStateAutomaton] :
    | elements=element* {TimedFiniteStateAutomaton(elements)}

element[Element] :
    | "state" name=ID ";" {State(name)}
    | "transition" "from" from=ID "to" to=ID trigger? ";"
      {Transition(from, to, trigger)}

trigger[Trigger] :
    | "on" event=ID {EventTrigger(event)}
    | "after" time=NUMBER {TimeoutTrigger(time)}


NUMBER  : /[+-]?([0-9]+(\.[0-9]*)?|\.[0-9]+)/
ID      : /[_a-zA-Z][_a-zA-Z0-9]*/
WS      : /[ \t\r\n]+/
COMMENT : /\s*\/\/[^\n]*/

@ignore(WS, COMMENT)
```

TDF Model

```
TimedFiniteStateAutomaton(elements: Element[])

interface Element()

State(name: tok): Element

Transition(from: tok, to: tok, trigger: Trigger?): Element

interface Trigger()
EventTrigger(event: tok): Trigger
TimeoutTrigger(time: tok): Trigger
```

Example Model: Power-window states

```
state neutral;
state down;
state up;
state emergency;

transition from initial   to neutral;
transition from neutral   to down      on press_down;
transition from down      to neutral   on release_down;
transition from neutral   to up        on press_up;
transition from up        to neutral   on release_up;
transition from up        to emergency on excessive_force;
transition from emergency to neutral   after 1;
```

Example Model: The Return

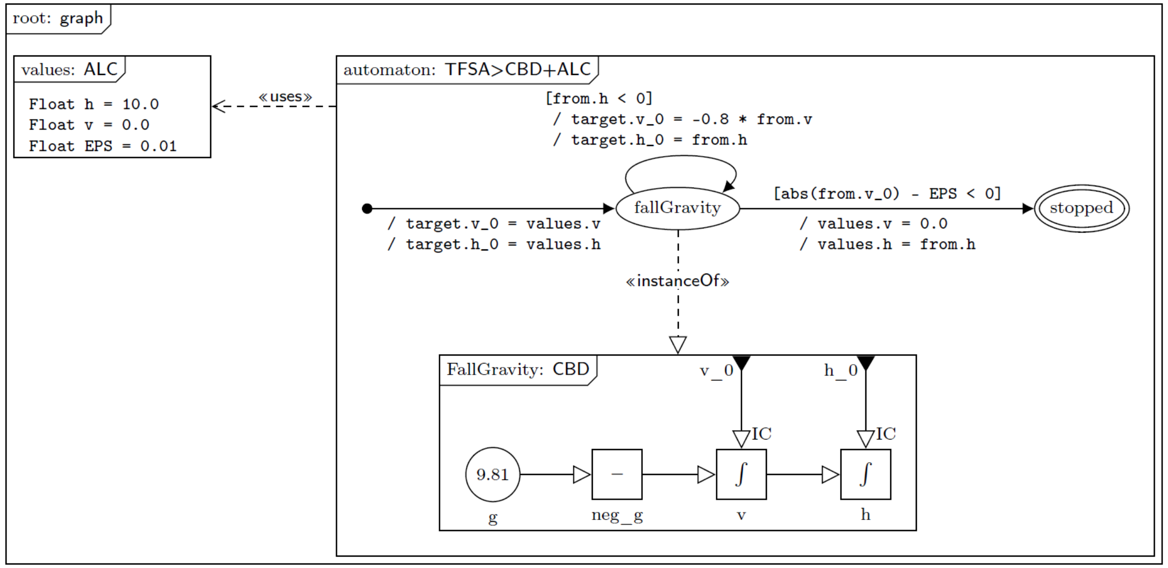
```
values = >alc#block{
    // ...
}
```

Dynamic language selection

```
automaton = >tfsa:pp{
    FallGravity = >cbd{
        parameter h_0;
        parameter v_0;
        // ...
    }
    initial state start;
    state fallGravity: FallGravity;
    final state stopped;

    // ...
    transition from fallGravity to fallGravity
        when [fallGravity.h < 0]
        do {
            target.v_0 = -0.8 * from.v
            target.h_0 = from.h
        };
    // ...
}
```

Static language selection

```
model_embed[embed] :
    | ">" @embed.select "{" @embed "}" {embed}
```

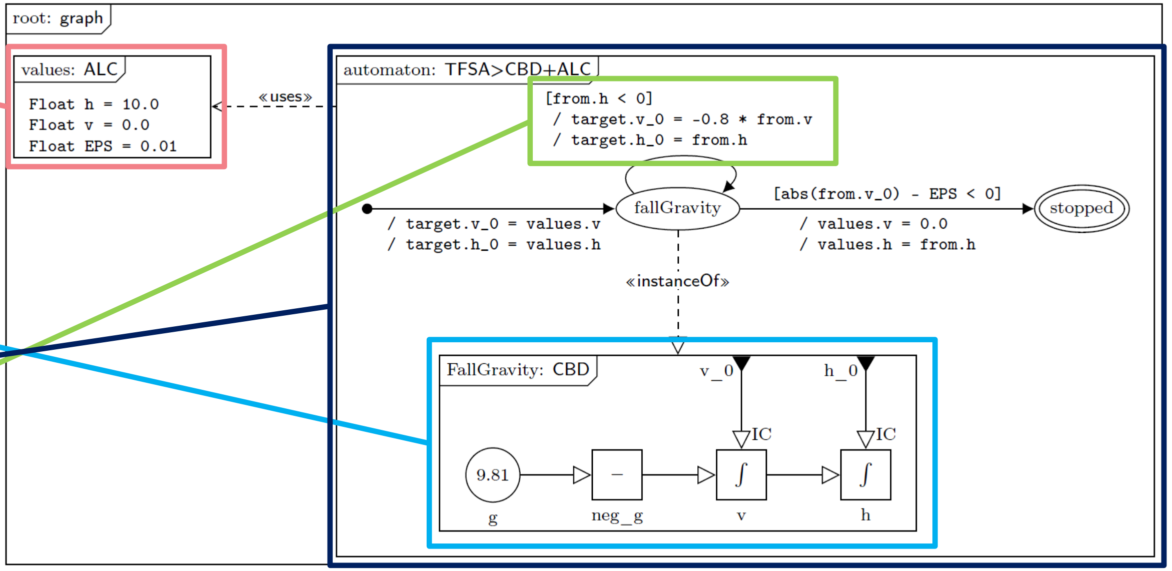


```
condition_embed[embed] :
    | "[" @embed("alc", "expression") "]" {embed}
```

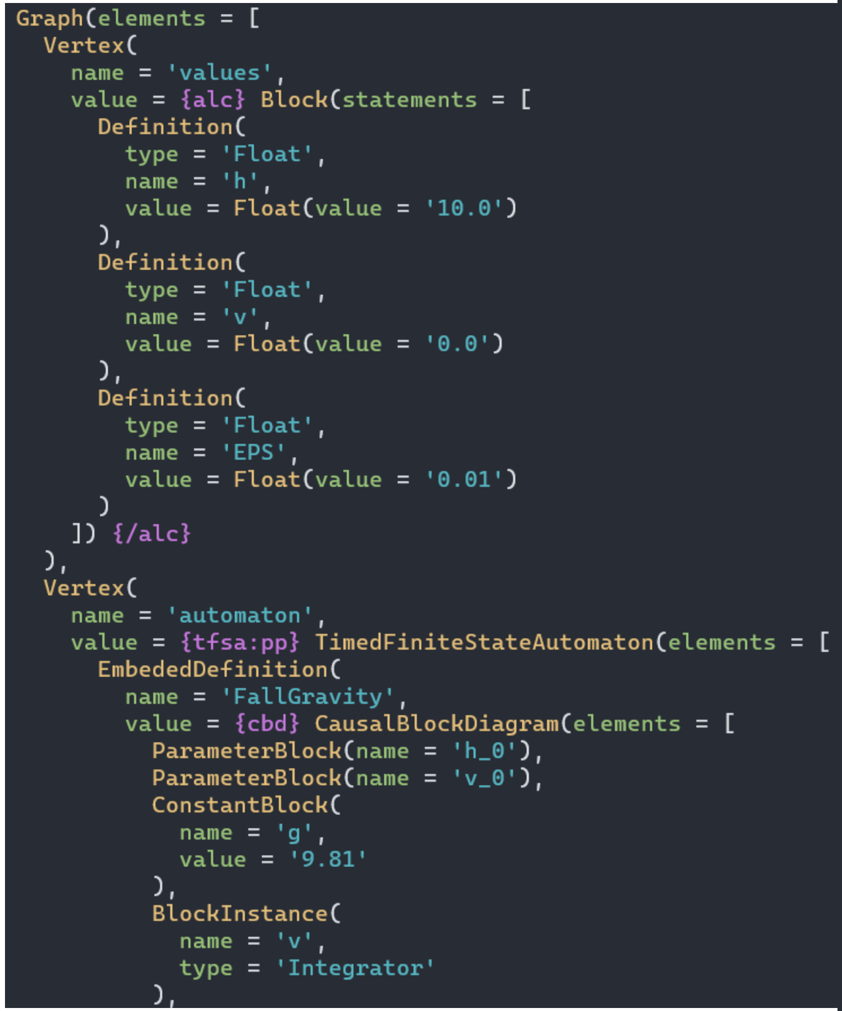
Example Model: The Return

```
values = >alc#block{
    // ...
}
```

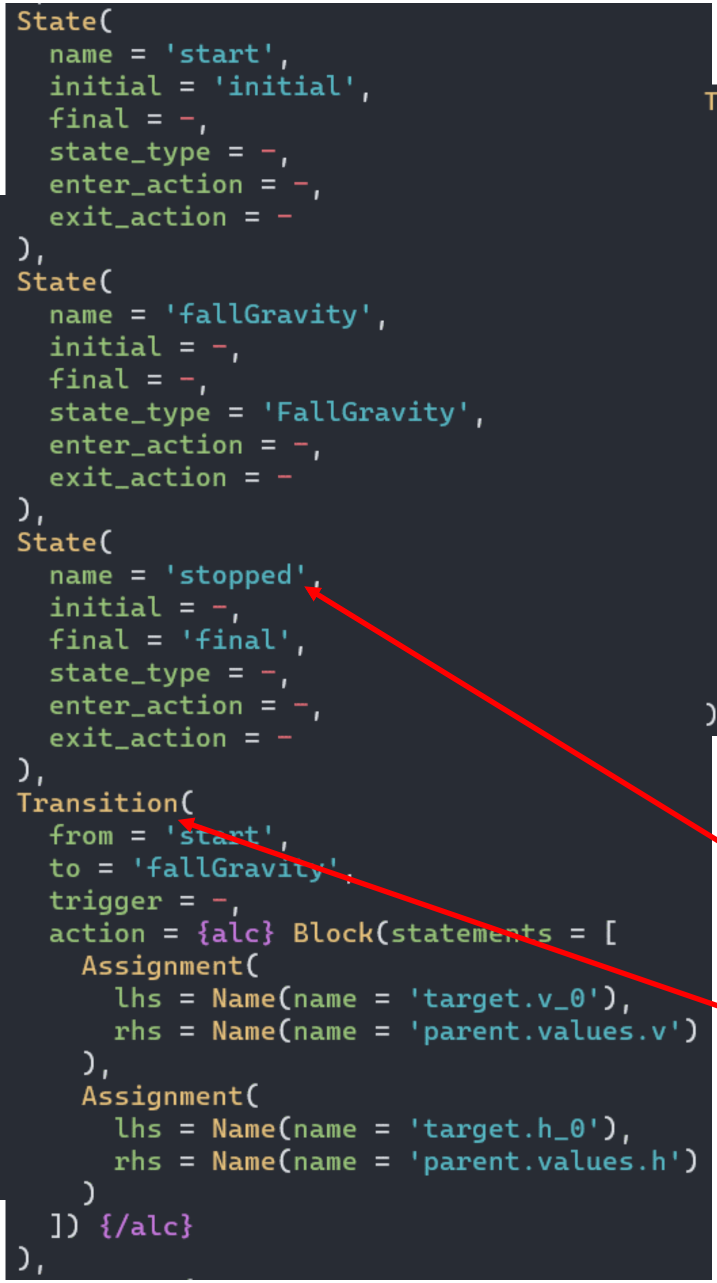
```
automaton = >tfsa:pp{
    FallGravity = >cbd{
        parameter h_0;
        parameter v_0;
        // ...
    }
    initial state start;
    state fallGravity: FallGravity;
    final state stopped;

    // ...
    transition from fallGravity to fallGravity
        when [fallGravity.h < 0]
        do {
            target.v_0 = -0.8 * from.v
            target.h_0 = from.h
        };
    // ...
}
```



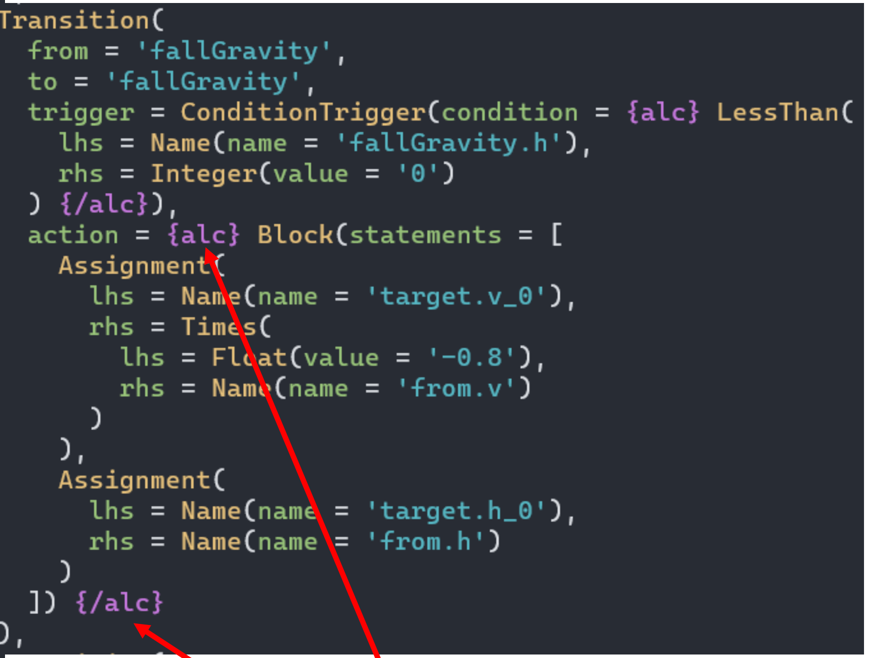
Demo



```
Graph(elements = [
  Vertex(
    name = 'values',
    value = {alc} Block(statements = [
      Definition(
        type = 'Float',
        name = 'h',
        value = Float(value = '10.0')
      ),
      Definition(
        type = 'Float',
        name = 'v',
        value = Float(value = '0.0')
      ),
      Definition(
        type = 'Float',
        name = 'EPS',
        value = Float(value = '0.01')
      )
    ]) {/alc}
  ),
  Vertex(
    name = 'automaton',
    value = {tfsa:pp} TimedFiniteStateAutomaton(elements = [
      EmbededDefinition(
        name = 'FallGravity',
        value = {cbd} CausalBlockDiagram(elements = [
          ParameterBlock(name = 'h_0'),
          ParameterBlock(name = 'v_0'),
          ConstantBlock(
            name = 'g',
            value = '9.81'
          ),
          BlockInstance(
            name = 'v',
            type = 'Integrator'
          ),
```



```
State(
  name = 'start',
  initial = 'initial',
  final = -,
  state_type = -,
  enter_action = -,
  exit_action = -
),
State(
  name = 'fallGravity',
  initial = -,
  final = -,
  state_type = 'FallGravity',
  enter_action = -,
  exit_action = -
),
State(
  name = 'stopped',
  initial = -,
  final = 'final',
  state_type = -,
  enter_action = -,
  exit_action = -
),
Transition(
  from = 'start',
  to = 'fallGravity',
  trigger = -,
  action = {alc} Block(statements = [
    Assignment(
      lhs = Name(name = 'target.v_0'),
      rhs = Name(name = 'parent.values.v')
    ),
    Assignment(
      lhs = Name(name = 'target.h_0'),
      rhs = Name(name = 'parent.values.h')
    )
  ]) {/alc}
),
```



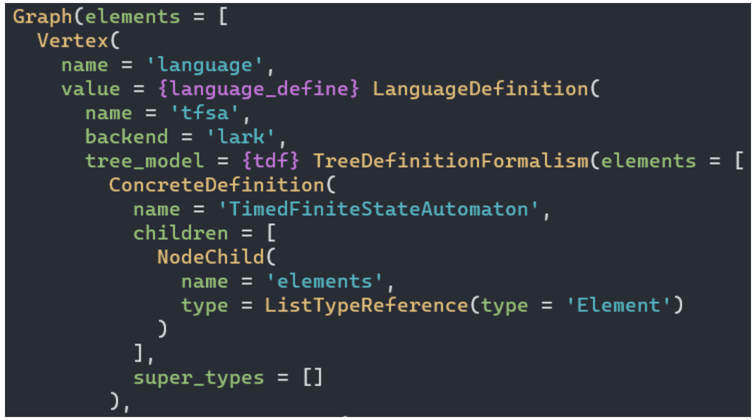
```
Transition(
  from = 'fallGravity',
  to = 'fallGravity',
  trigger = ConditionTrigger(condition = {alc} LessThan(
    lhs = Name(name = 'fallGravity.h'),
    rhs = Integer(value = '0')
  ) {/alc}),
  action = {alc} Block(statements = [
    Assignment(
      lhs = Name(name = 'target.v_0'),
      rhs = Times(
        lhs = Float(value = '-0.8'),
        rhs = Name(name = 'from.v')
      )
    ),
    Assignment(
      lhs = Name(name = 'target.h_0'),
      rhs = Name(name = 'from.h')
    )
  ]) {/alc}
),
```

Embedded fragment indicators

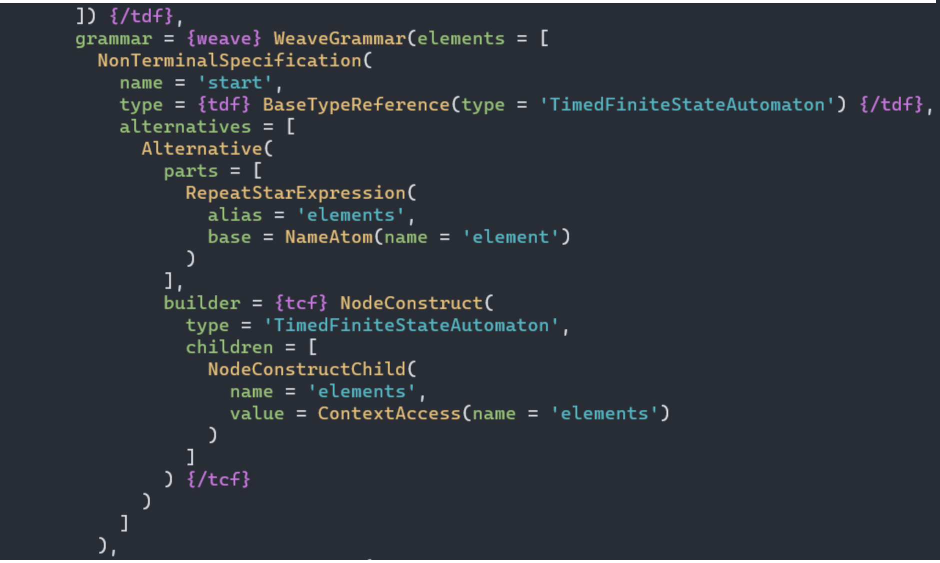
TokenInstance

ConcreteInstance

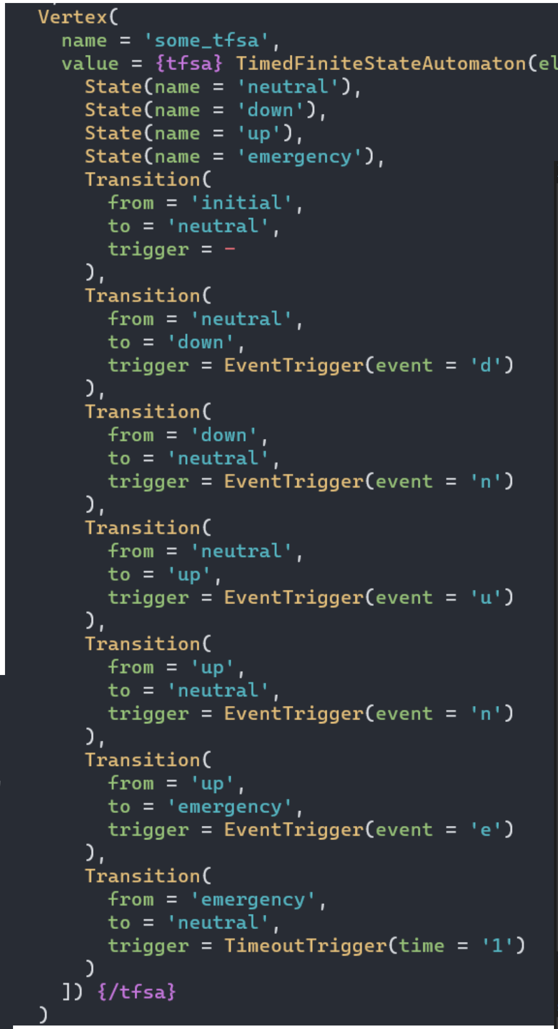
Demo



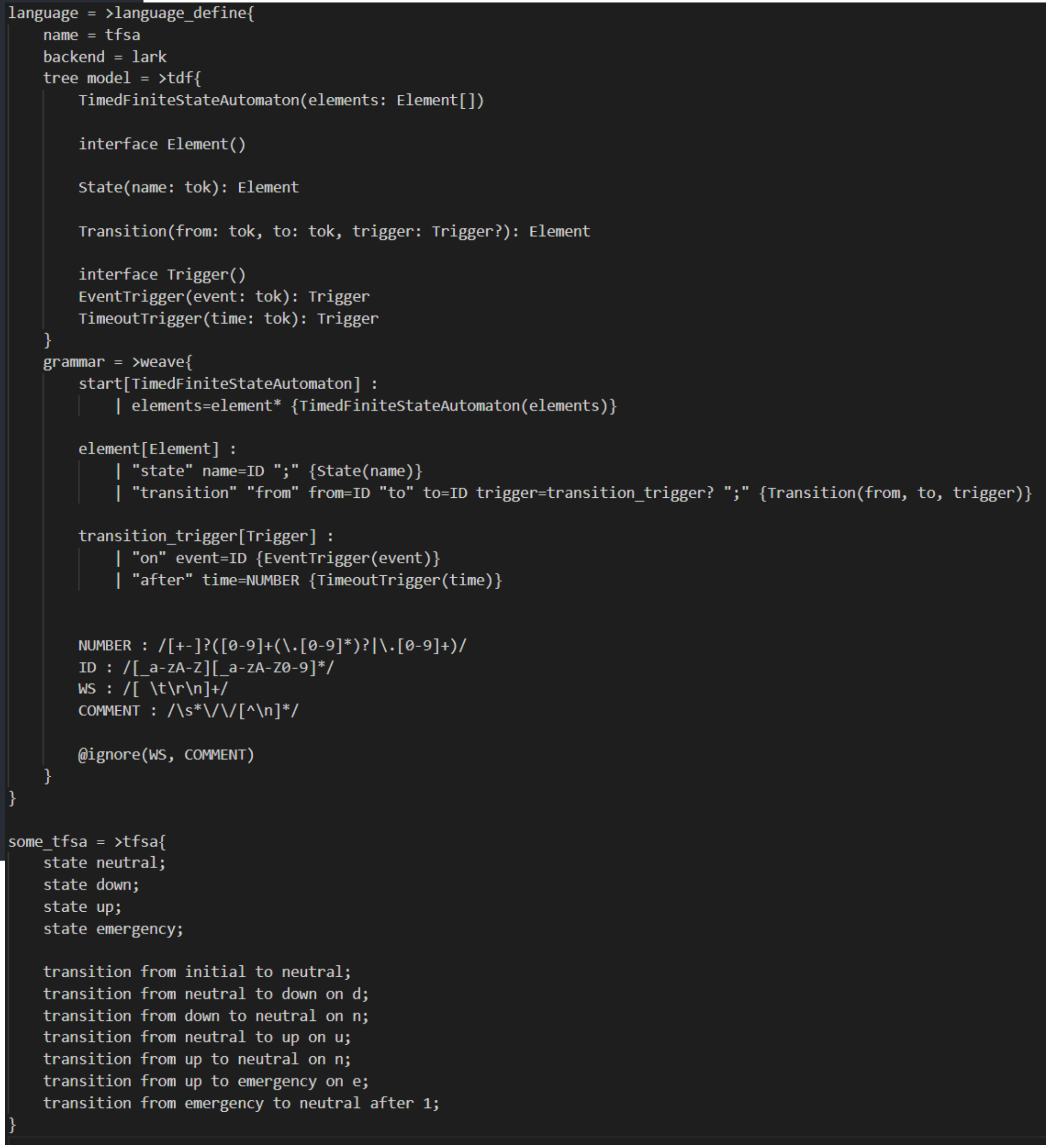
```
Graph(elements = [
  Vertex(
    name = 'language',
    value = {language_define} LanguageDefinition(
      name = 'tfsa',
      backend = 'lark',
      tree_model = {tdf} TreeDefinitionFormalism(elements = [
        ConcreteDefinition(
          name = 'TimedFiniteStateAutomaton',
          children = [
            NodeChild(
              name = 'elements',
              type = ListTypeReference(type = 'Element')
            )
          ],
          super_types = []
        ),
```



```
]) {/tdf},
grammar = {weave} WeaveGrammar(elements = [
  NonTerminalSpecification(
    name = 'start',
    type = {tdf} BaseTypeReference(type = 'TimedFiniteStateAutomaton') {/tdf},
    alternatives = [
      Alternative(
        parts = [
          RepeatStarExpression(
            alias = 'elements',
            base = NameAtom(name = 'element')
          )
        ],
        builder = {tcf} NodeConstruct(
          type = 'TimedFiniteStateAutomaton',
          children = [
            NodeConstructChild(
              name = 'elements',
              value = ContextAccess(name = 'elements')
            )
          ]
        ) {/tcf}
      )
    ]
  ),
```



```
Vertex(
  name = 'some_tfsa',
  value = {tfsa} TimedFiniteStateAutomaton(elements = [
    State(name = 'neutral'),
    State(name = 'down'),
    State(name = 'up'),
    State(name = 'emergency'),
    Transition(
      from = 'initial',
      to = 'neutral',
      trigger = -
    ),
    Transition(
      from = 'neutral',
      to = 'down',
      trigger = EventTrigger(event = 'd')
    ),
    Transition(
      from = 'down',
      to = 'neutral',
      trigger = EventTrigger(event = 'n')
    ),
    Transition(
      from = 'neutral',
      to = 'up',
      trigger = EventTrigger(event = 'u')
    ),
    Transition(
      from = 'up',
      to = 'neutral',
      trigger = EventTrigger(event = 'n')
    ),
    Transition(
      from = 'up',
      to = 'emergency',
      trigger = EventTrigger(event = 'e')
    ),
    Transition(
      from = 'emergency',
      to = 'neutral',
      trigger = TimeoutTrigger(time = '1')
    )
  ]) {/tfsa}
)
```



```
language = >language_define{
    name = tfsa
    backend = lark
    tree model = >tdf{
        TimedFiniteStateAutomaton(elements: Element[])

        interface Element()

        State(name: tok): Element

        Transition(from: tok, to: tok, trigger: Trigger?): Element

        interface Trigger()
        EventTrigger(event: tok): Trigger
        TimeoutTrigger(time: tok): Trigger
    }
    grammar = >weave{
        start[TimedFiniteStateAutomaton] :
            | elements=element* {TimedFiniteStateAutomaton(elements)}

        element[Element] :
            | "state" name=ID ";" {State(name)}
            | "transition" "from" from=ID "to" to=ID trigger=transition_trigger? ";" {Transition(from, to, trigger)}

        transition_trigger[Trigger] :
            | "on" event=ID {EventTrigger(event)}
            | "after" time=NUMBER {TimeoutTrigger(time)}


        NUMBER : /[+-]?([0-9]+(\.[0-9]*)?|\.[0-9]+)/
        ID : /[_a-zA-Z][_a-zA-Z0-9]*/
        WS : /[ \t\r\n]+/
        COMMENT : /\s*\/\/[^\n]*/

        @ignore(WS, COMMENT)
    }
}

some_tfsa = >tfsa{
    state neutral;
    state down;
    state up;
    state emergency;

    transition from initial to neutral;
    transition from neutral to down on d;
    transition from down to neutral on n;
    transition from neutral to up on u;
    transition from up to neutral on n;
    transition from up to emergency on e;
    transition from emergency to neutral after 1;
}
```

Conclusion

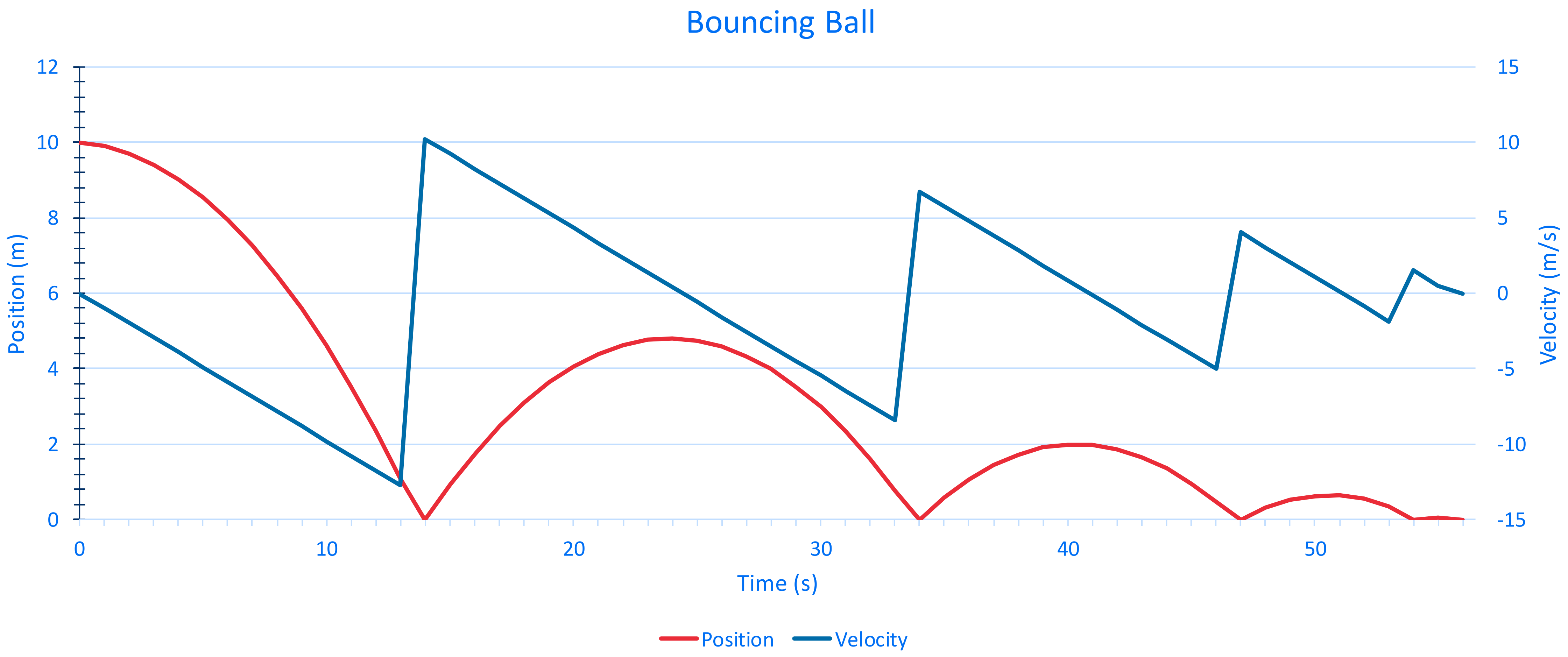
- **Goals achieved:**
  - Multi-language parsing
  - Dynamically selecting languages
  - Dynamically defining languages, and then using them
- **Goals not achieved:**
  - Exporting to the Modelverse
    - But, investigated and prototyped during research project
- **Future work:**
  - Language Server (using Language Server Protocol)
  - Blackbox parsers

Thank you for your time

References

- [1] R. Paredis, J. Denil, H. Vangheluwe, “Specifying and Executing the Combination of Timed Finite State Automata and Causal-block Diagrams by Mapping Onto DEVS”, 2021
- [2] A. Aho, R. Sethi, J. Ullman, “Compilers: Principles, Techniques & Tools”, 1986

Example Model: The Maths


Bouncing Ball
Position (m)
Velocity (m/s)
Time (s)
Position
Velocity