simularity

SimGen

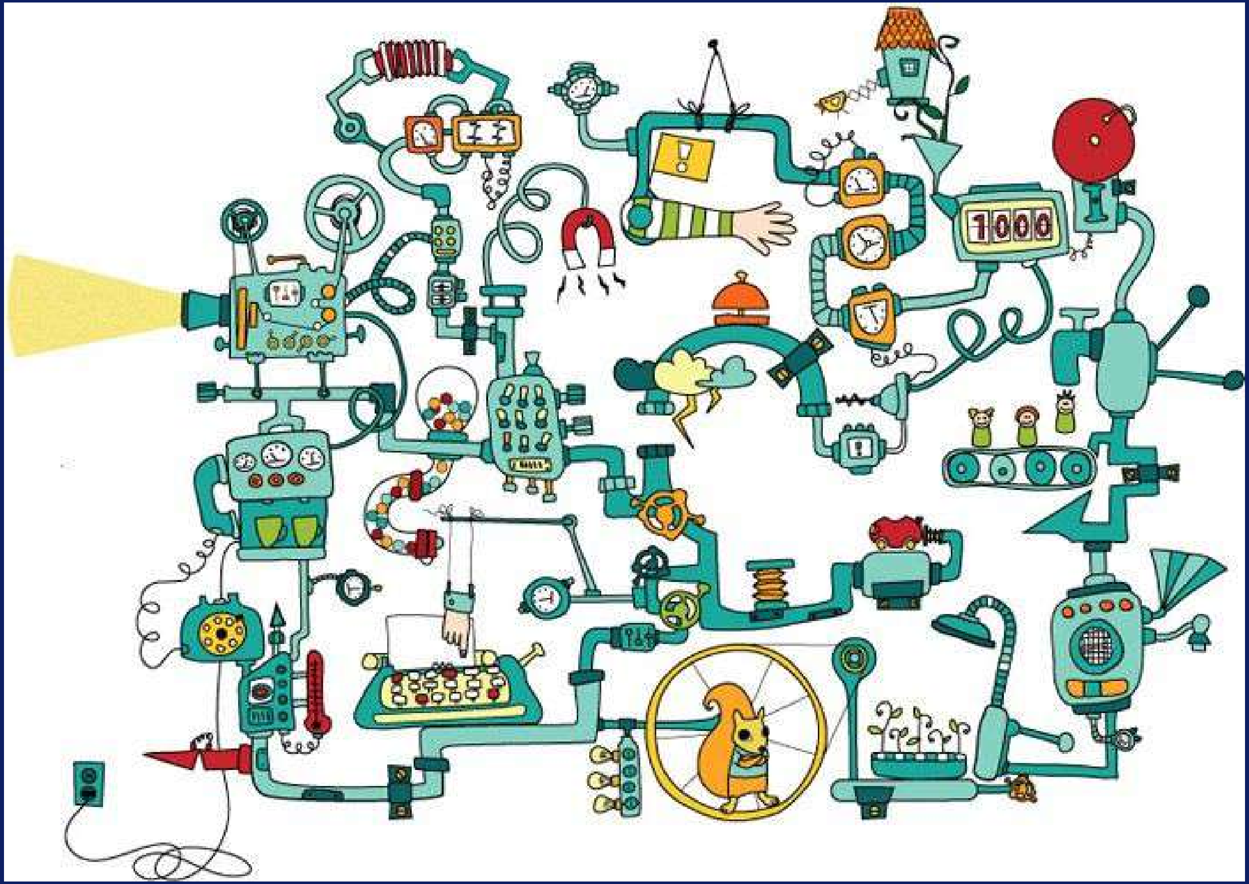
A New Simulation Language

Anomalometry

simularity

Simulation

simularity

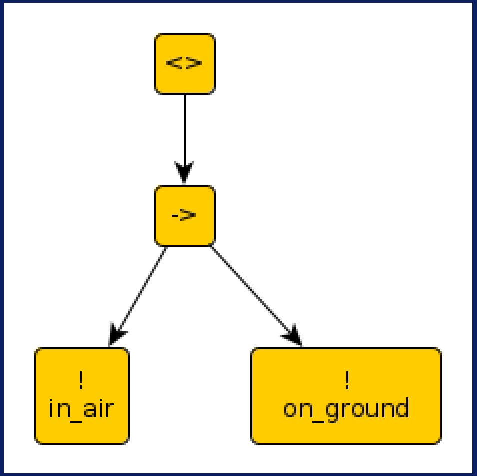


Behaviours

simularity

Behaviour is a pattern of activity continuing over time

Bouncing Ball



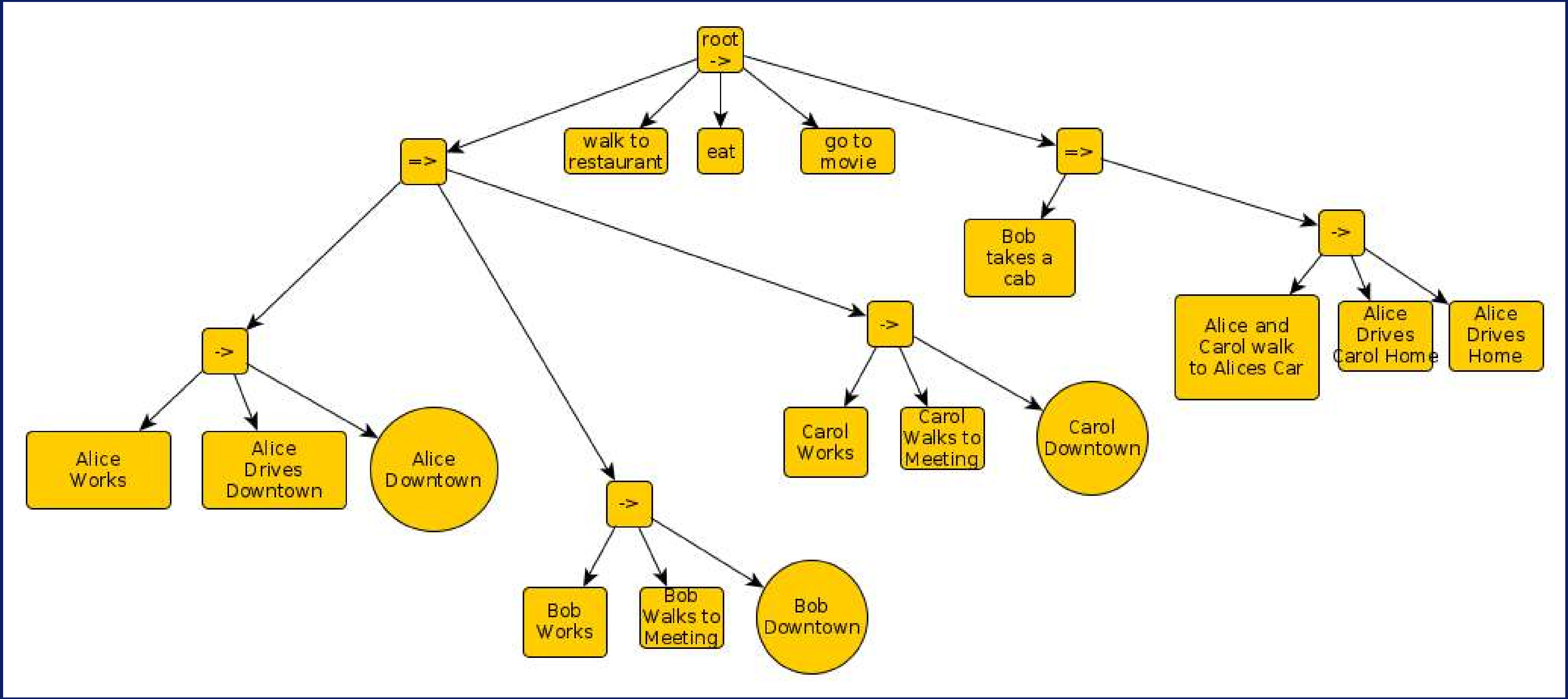
Examples

simularity

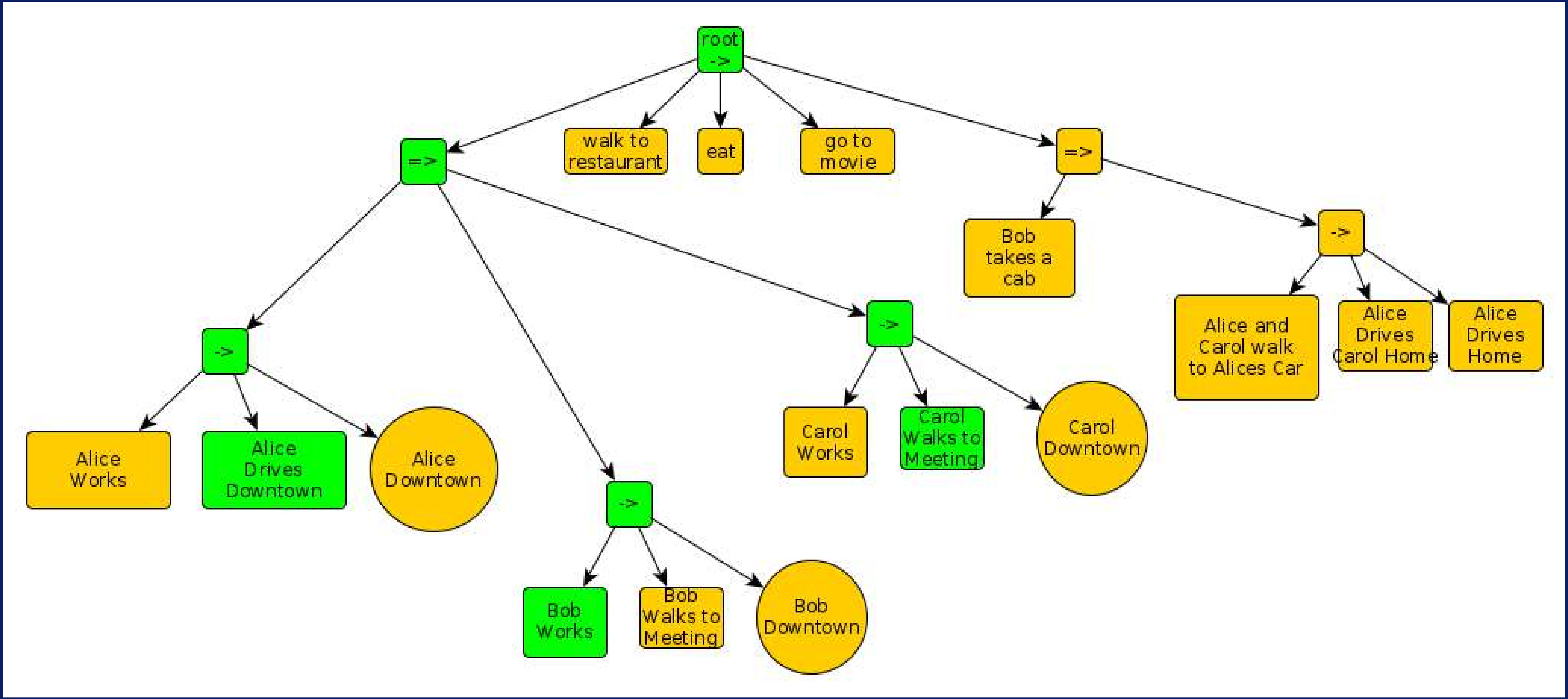
- Non-player character for a game
- Artificial agent
- System simulation
- System control (eg pre-launch countdown)
- Progress through a process (eg website signup)

Evening Out

simularity

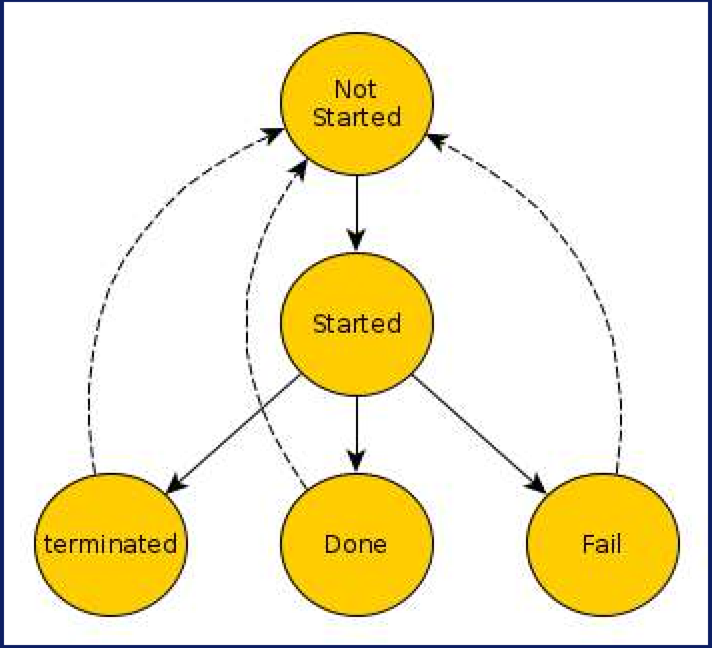


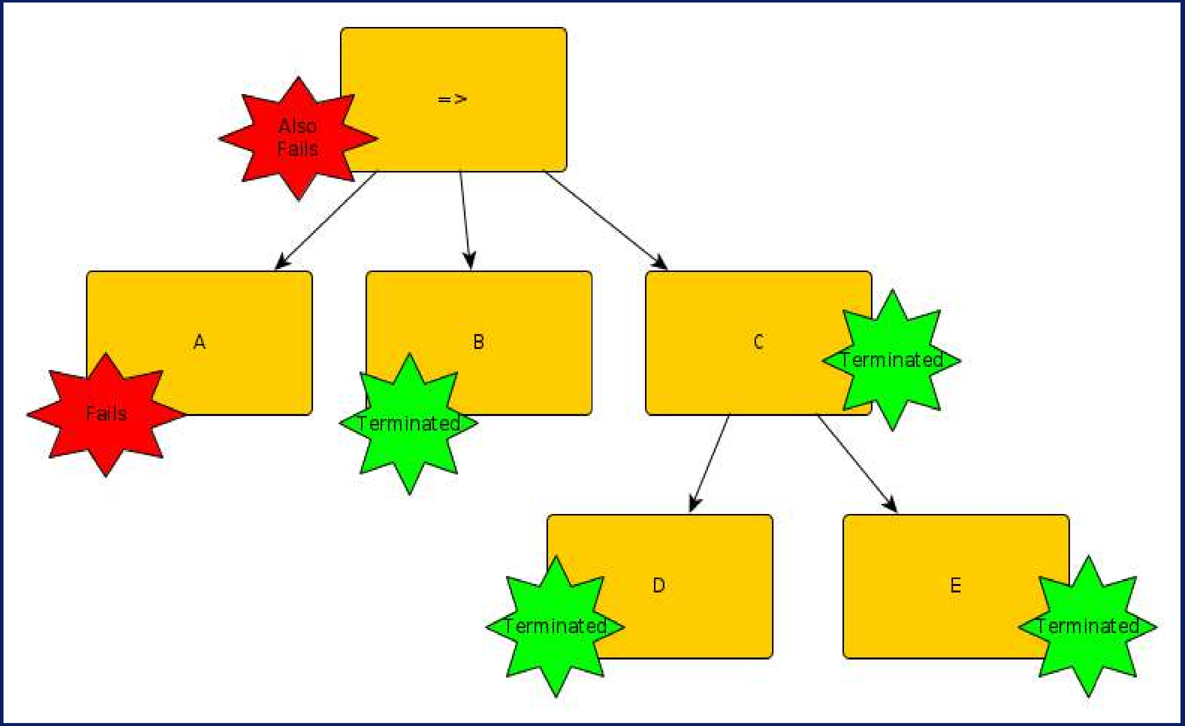
Running

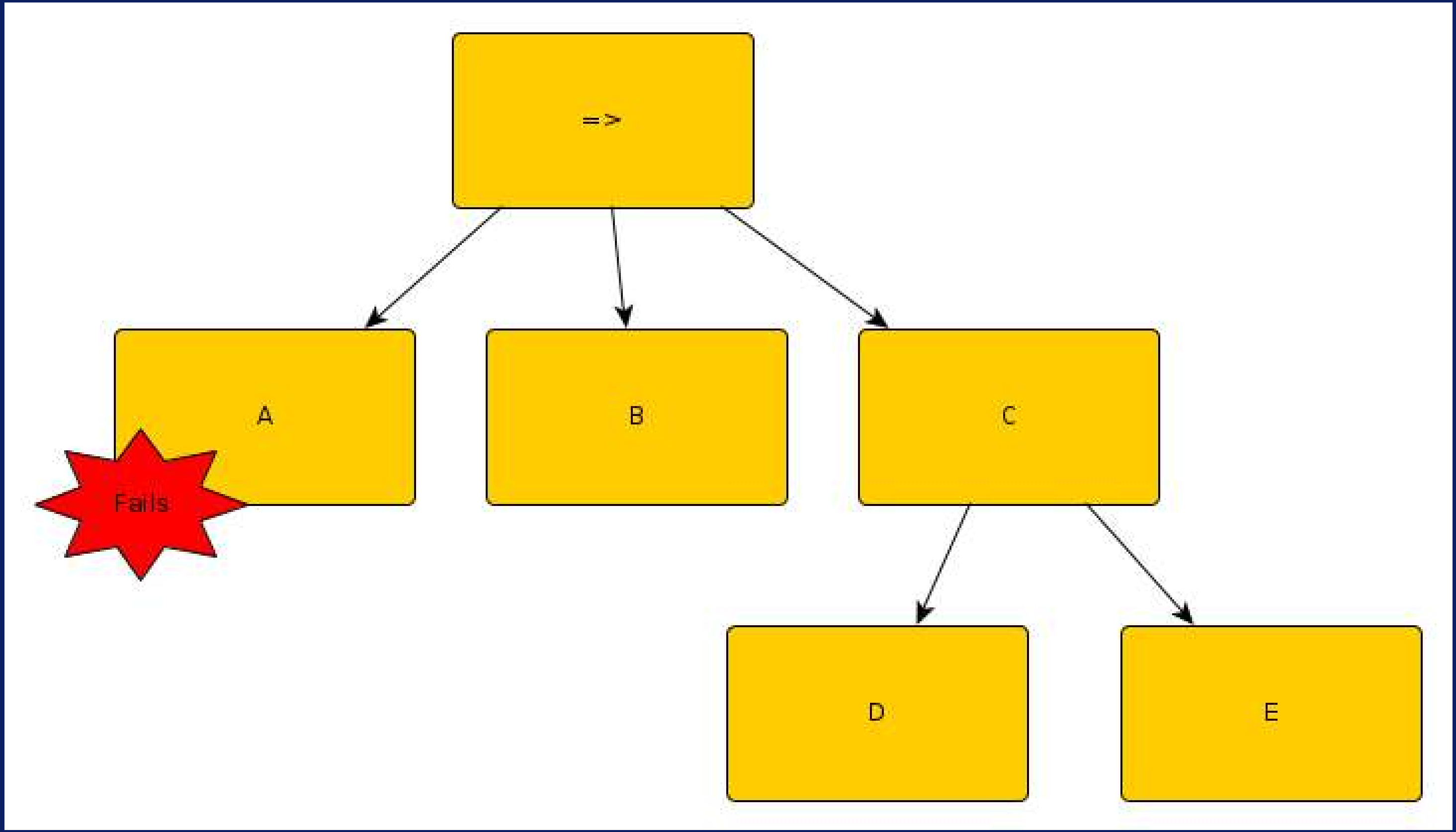


root ->
=>
walk to restaurant
eat
go to movie
=>
->
Alice Works
Alice Drives Downtown
Alice Downtown
->
Bob Works
Bob Walks to Meeting
Bob Downtown
->
Carol Works
Carol Walks to Meeting
Carol Downtown
Bob takes a cab
->
Alice and Carol walk to Alices Car
Alice Drives Carol Home
Alice Drives Home

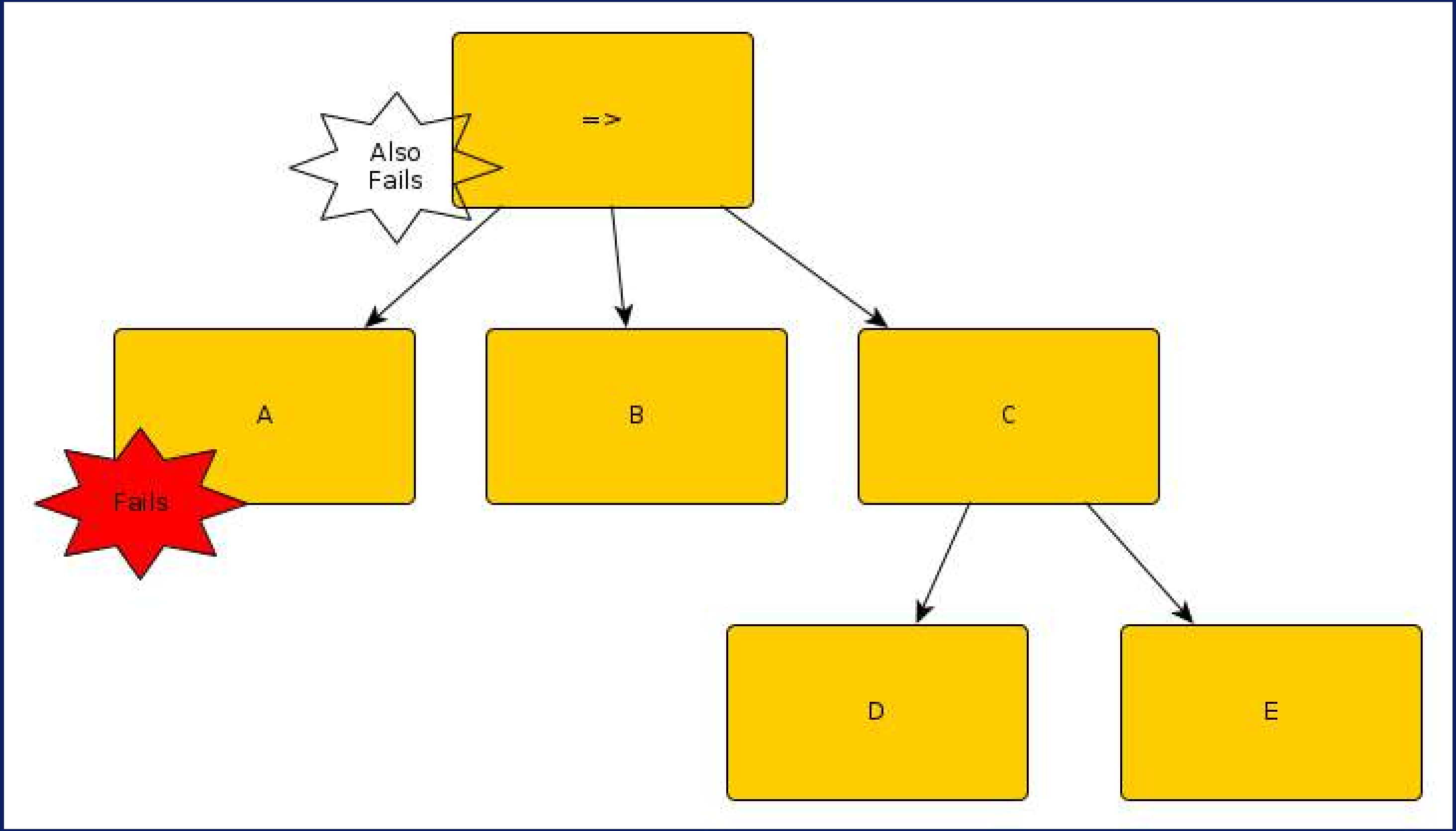
Life Cycle


Not Started
Started
terminated
Done
Fail

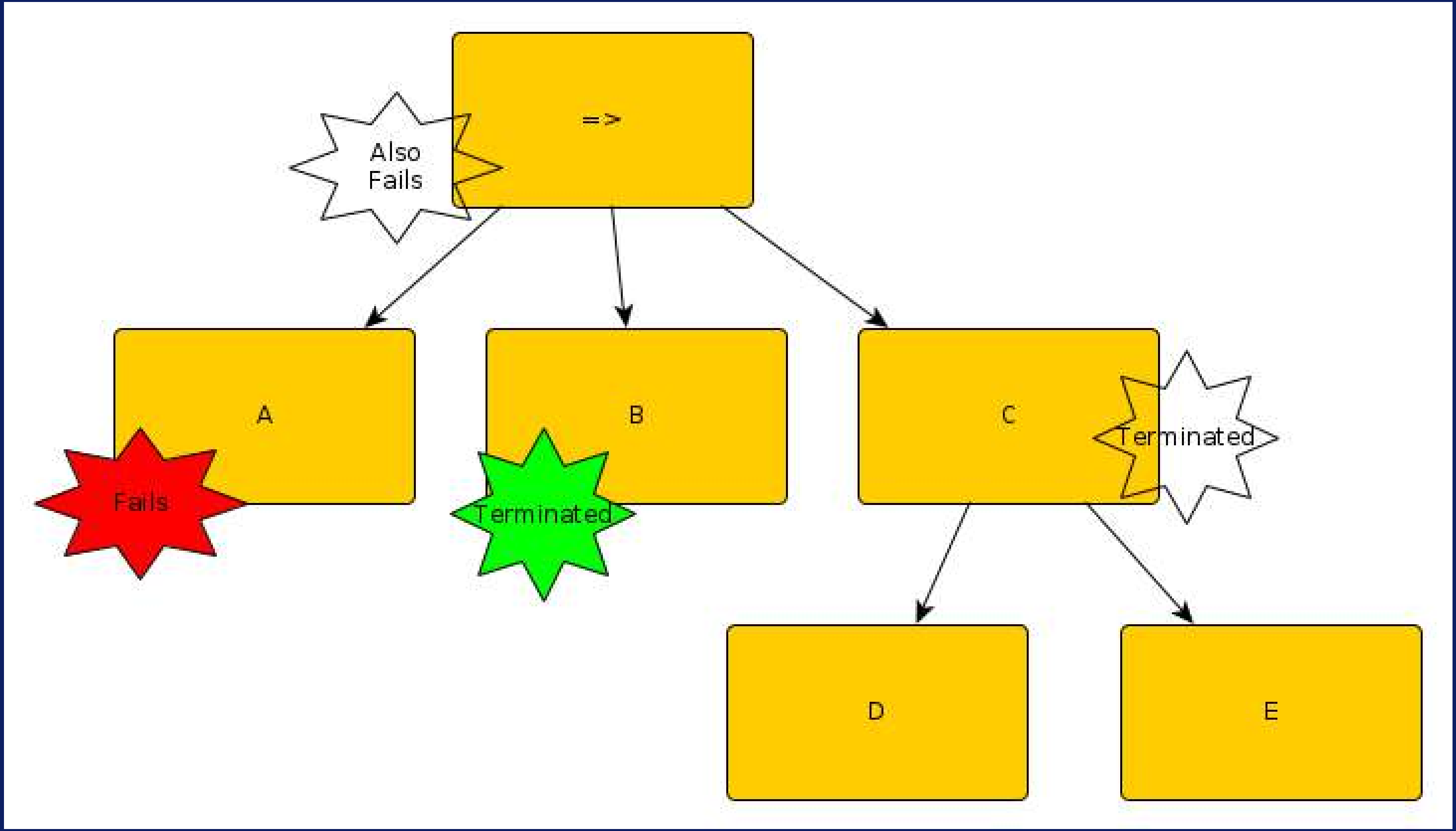

=>
Also Fails
A
B
C
Fails
Terminated
Terminated
D
E
Terminated
Terminated

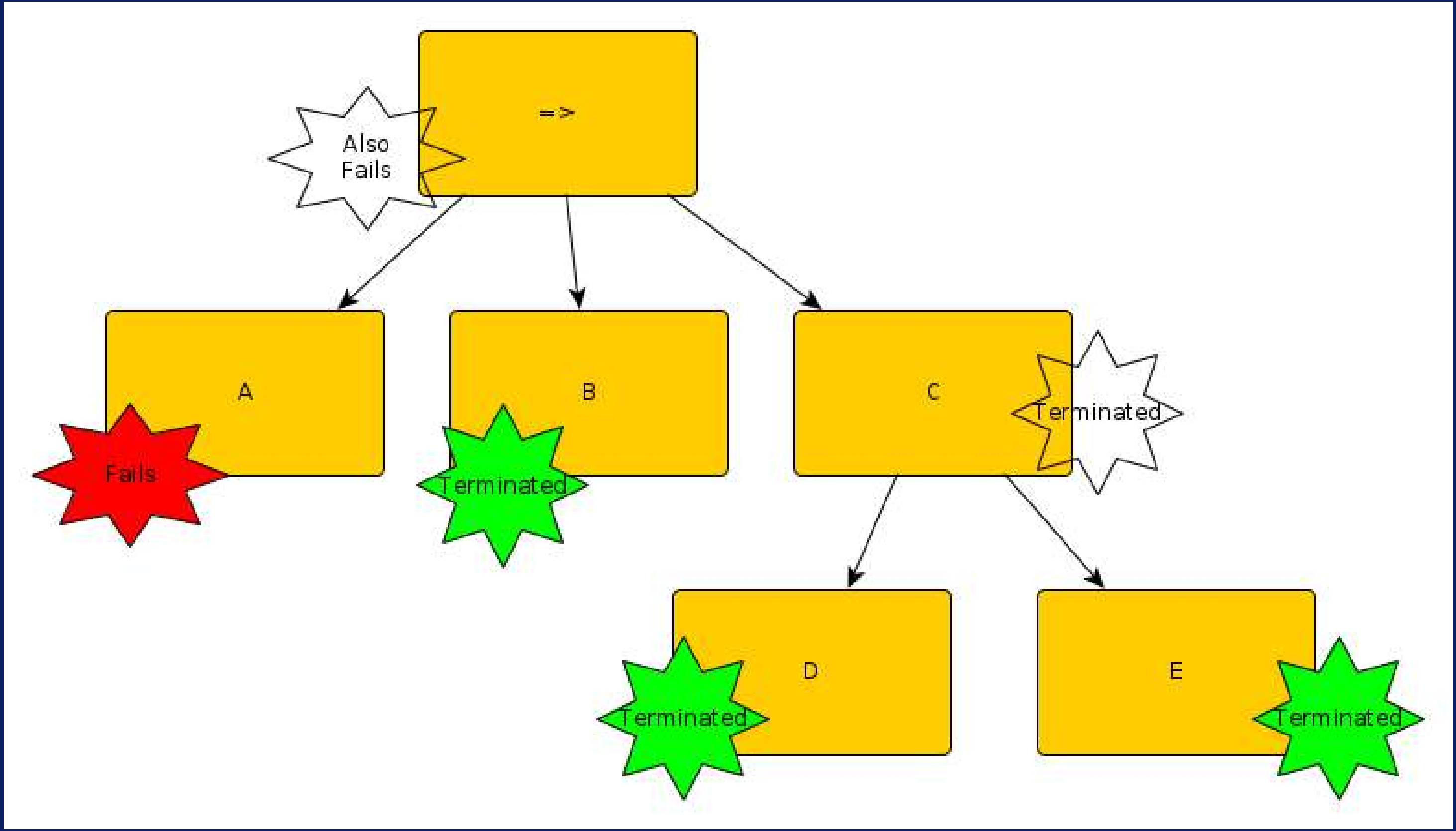


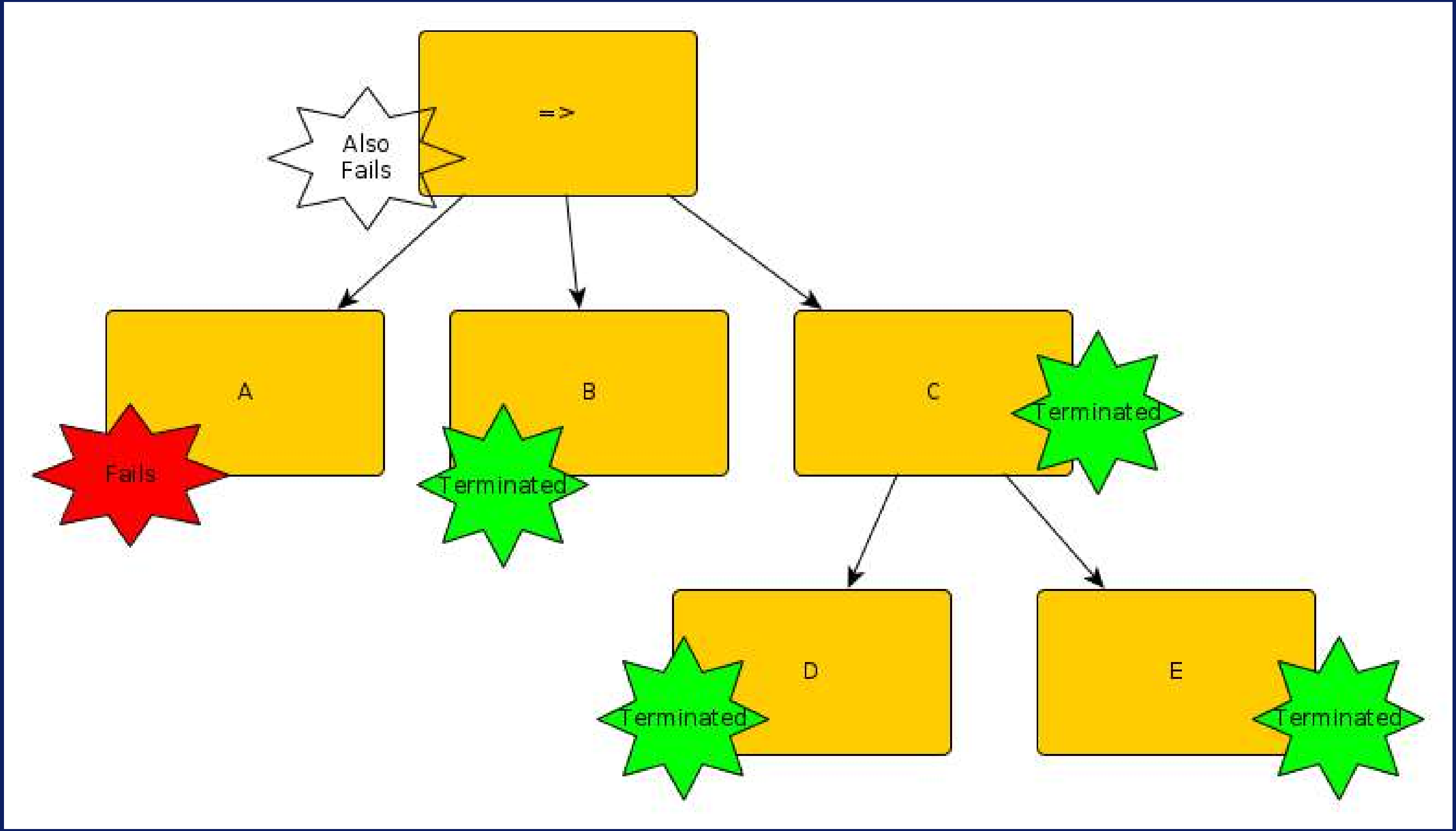
=>
A
B
C
Fails
D
E

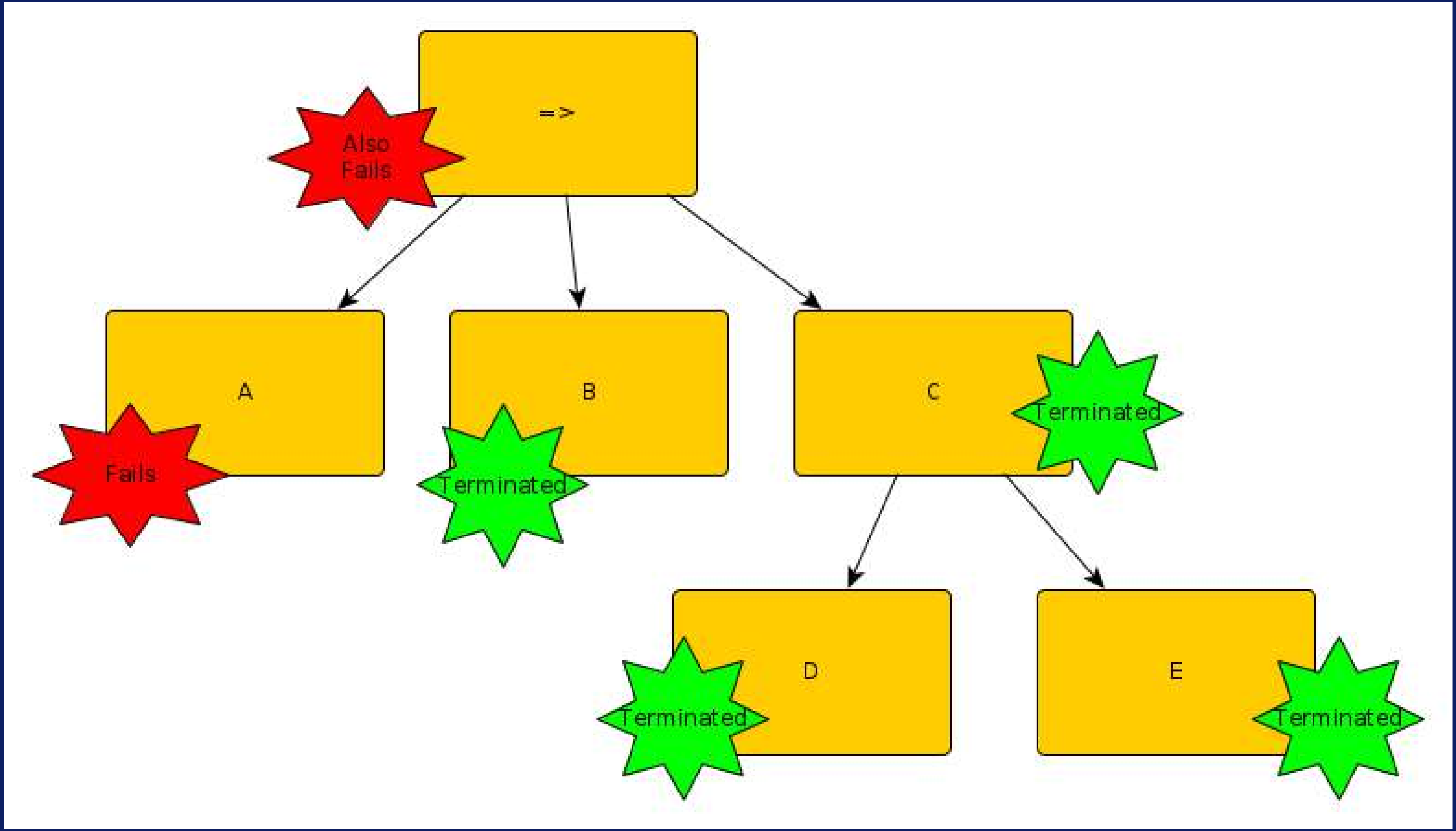

=>
Also Fails
A
Fails
B
C
D
E

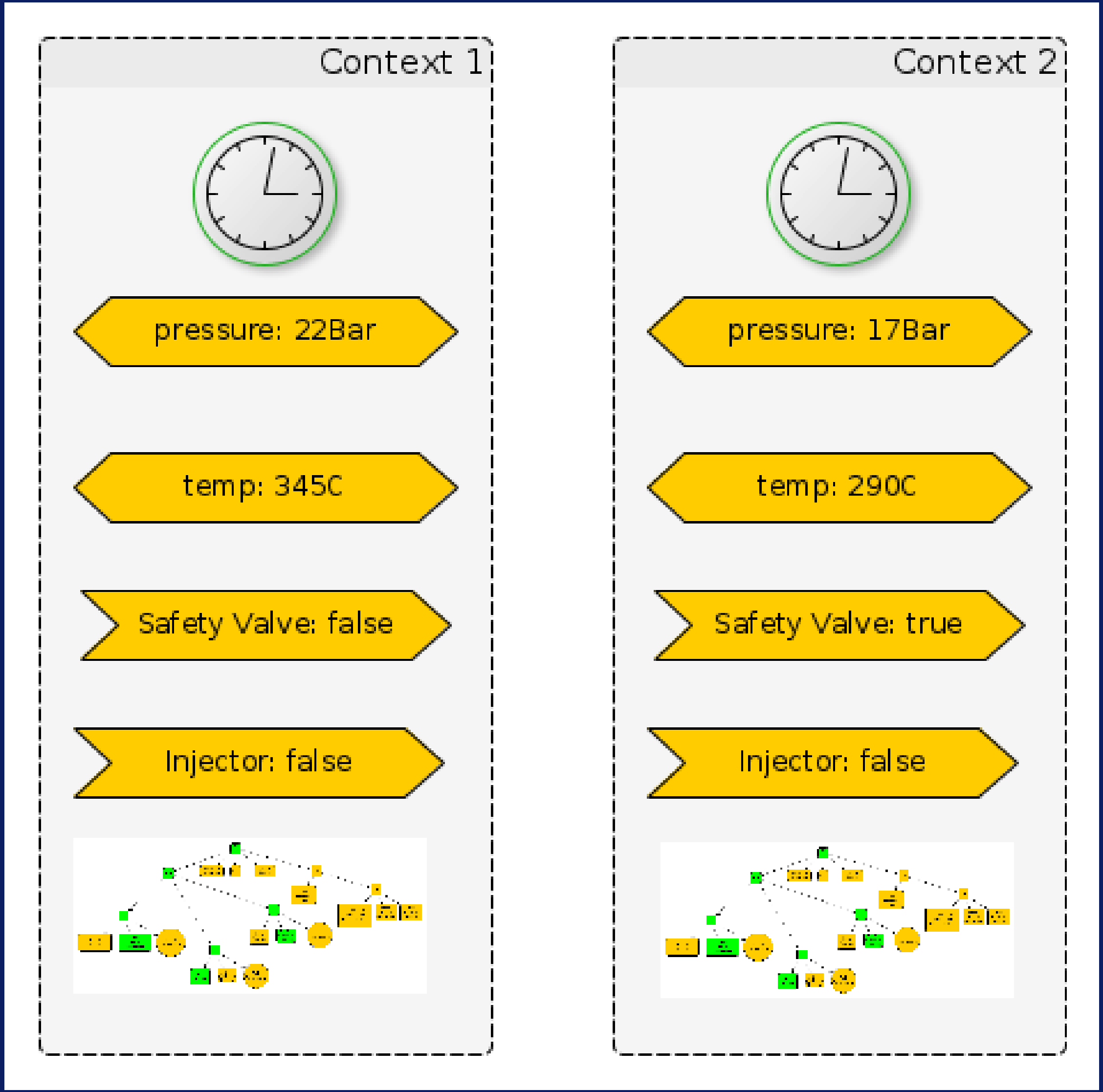
simularity




=>
Also Fails
A
Fails
B
Terminated
C
Terminated
D
Terminated
E
Terminated


=>
Also Fails
A
Fails
B
Terminated
C
Terminated
D
Terminated
E
Terminated


=>
Also Fails
A
Fails
B
Terminated
C
Terminated
D
Terminated
E
Terminated


Context 1
pressure: 22Bar
temp: 345C
Safety Valve: false
Injector: false
Context 2
pressure: 17Bar
temp: 290C
Safety Valve: true
Injector: false

simularity

Consult the bt language file

use_bt('myprog.bt')

Syntax

simularity

```
name operator stuff .

{ operator stuff }


% comment

/* comment */
```

```
root <> { ->
          {! ydot = 1, y = 10
          ;; 1 > 7 },
          in_air,
          bounce
        }.

in_air !
    y = 0
   ; ydot := ydot – 0.1, y = y + ydot
   ; y >= 0
.

bounce !
   ydot = abs(ydot) * 0.9, y = 0
  ;; 1 > 7
```

simularity

Node Types

```
~? [child | weight ":" child]+
```

Randomly run a child. Default weight 1.0

```
-> child+
```

Do a sequence of things. If one fails, the node fails.

```
~> child+
```

Randomly order the children, and then execute as ->

simularity

=> child+

Run in parallel. If any fail, fail. Join at end.

Guard a condition. Enforce coordinated action.

=? child+

Run in parallel. If any fail, fail. If any succeed, succeed.

? condition

Check guard - checks the condition every tick. If the condition is false, it fails. If true, it succeeds

-? condition

Wait guard - waits until the condition is true and succeeds

set condition

makes the condition true

clear condition

makes the condition false

try child

run the child and succeed whether the child succeeds or fails

fail

just fail

not child

fail when child succeeds, succeed if child fails

dur number

wait this number of user time units, then succeed

pin child

emit a Simularity specific pair of 'pin' events

<> child

loop - run the child repeatedly until it fails

<--> child

retry loop - run the child repeatedly until it succeeds

PDQ Operator

simularity

- ! partial differential equation
- ! First_tick ; Rest_ticks ; Conds
- Cond tested at bottom
- := from last cycle
- = dataflow

PDQ = is + these

simularity

levy_flight(Prev, Lo, Hi) which performs a Levy Flight between its low and high values.

wander(Prev, Lo, Hi, Dist) randomly wanders a uniform 0-Dist on each step

clock() returns the current context clock

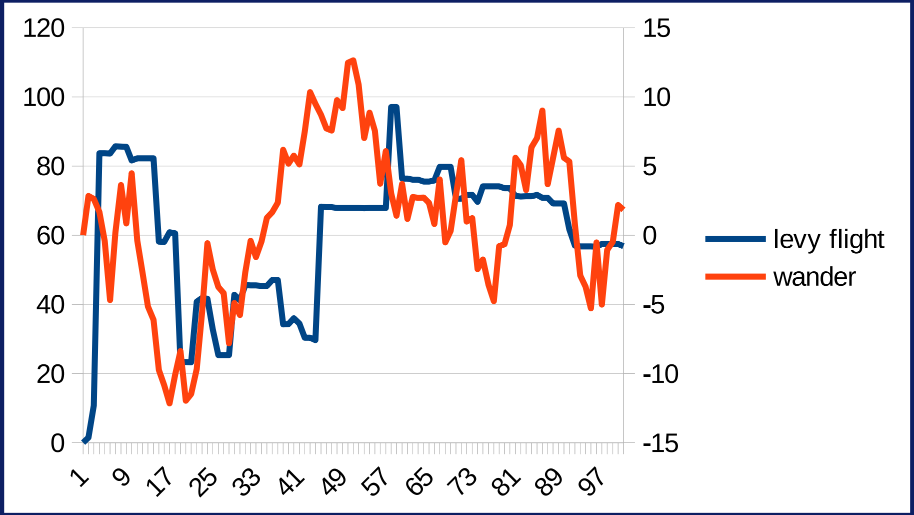
pow(Old, Exp) - exponential

lshift(Old, Bits) - left shift

rshift(Old, Bits) - right shift

bitor(A, B) - bitwise OR

bitand(A, B) - bitwise AND



Interacting with Prolog

Running

```
start_simulation(
        0,                  % the start time, in 'our' units
        60_000_000_000,     % how long our units are in nanos
        1,                  % how long a tick is in our units
        Extern)             % info passed with the

start_context(Root, Context, Time).
end_simulation
```

Messages

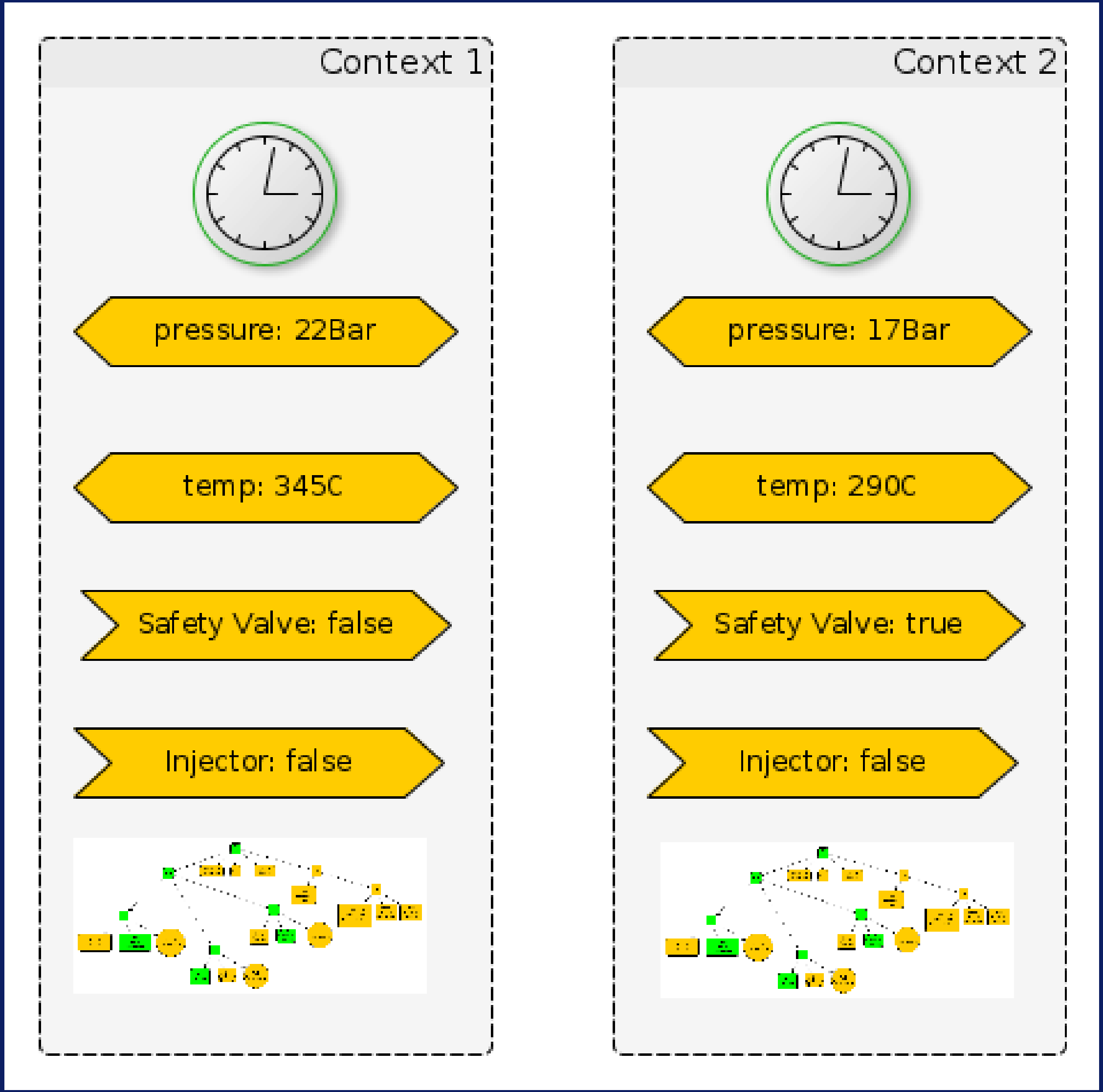
simularity

```
:- use_module(library(broadcast)).
:- listen(
     reading(Time, ContextTime,
               Context, Type, Value),
     handler(....)).
```

Messages

- reading(Time, ContextTime, Context, Name, Val)
- starting(Context-Type)
- stopped(Context-Type, Why)
- Why is done, fail, or terminated
- tick(Extern, Tick, NewExtern)

simularity


Context 1
pressure: 22Bar
temp: 345C
Safety Valve: false
Injector: false
Context 2
pressure: 17Bar
temp: 290C
Safety Valve: true
Injector: false

Future

- Variables are vectors
- Real PDQ
- Drop special syntax

Thanks

- Liz Derr at Simularity
- Ray Richardson at Simularity

Code available at

https://github.com/simularity/SimGen