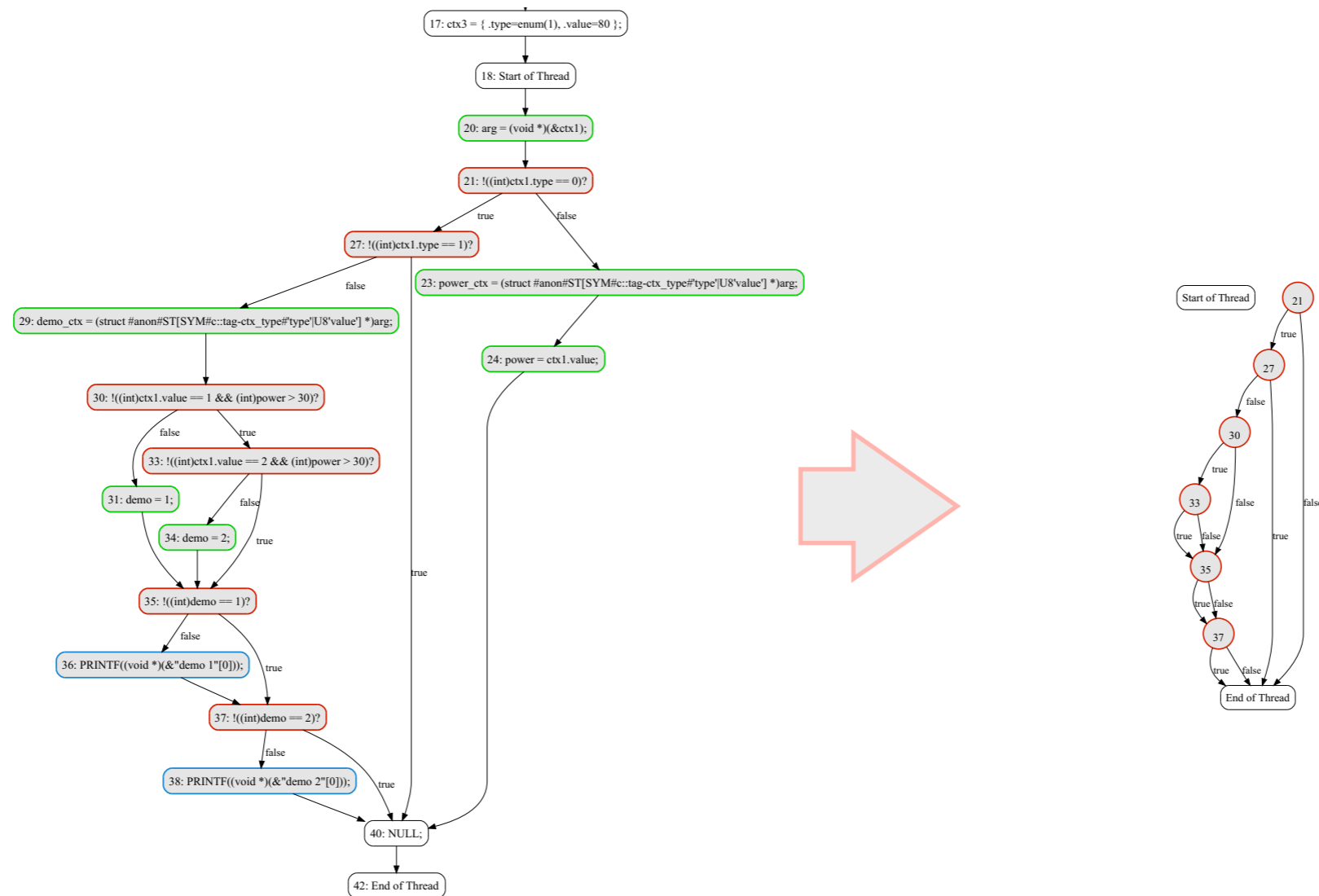
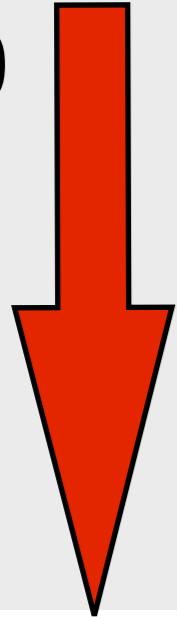


Model Extraction from Context-Aware Code

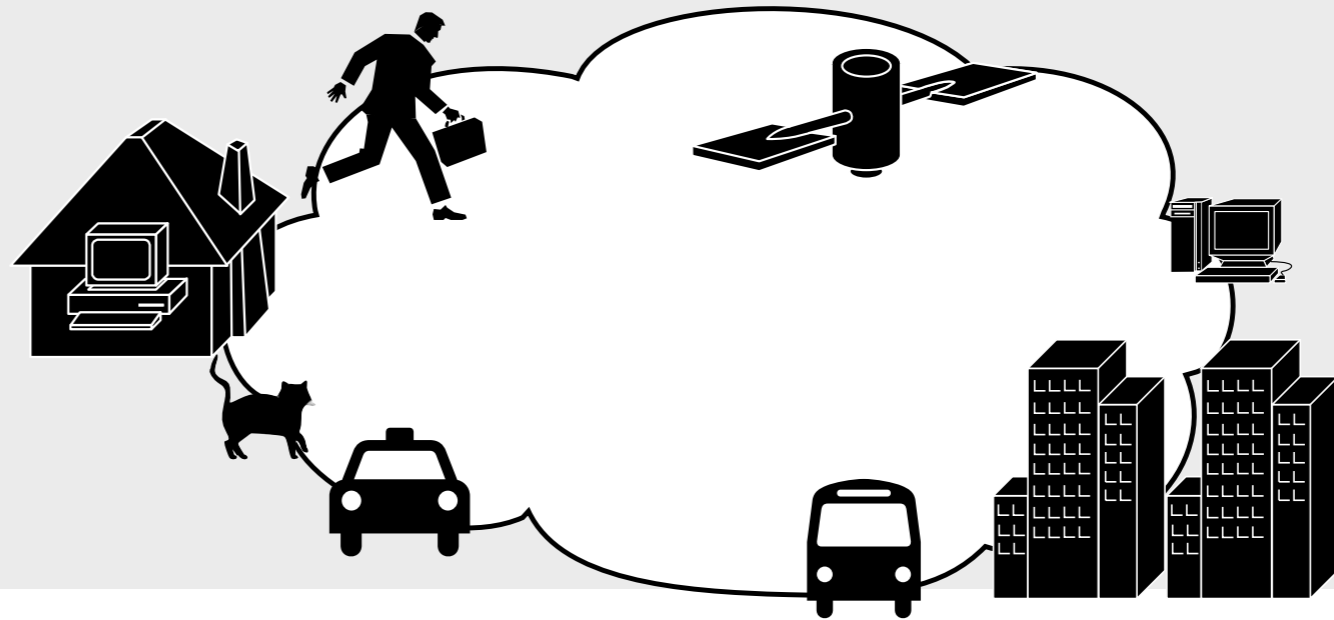


Ubiquitous Computing

2020



ubiquitous
computing



- Paradigm:
 - Interactions with **networked data-processing** systems available everywhere.
 - ...which autonomously **monitor and adapt** themselves and the environment.

Concept: Context Awareness

- Context awareness:

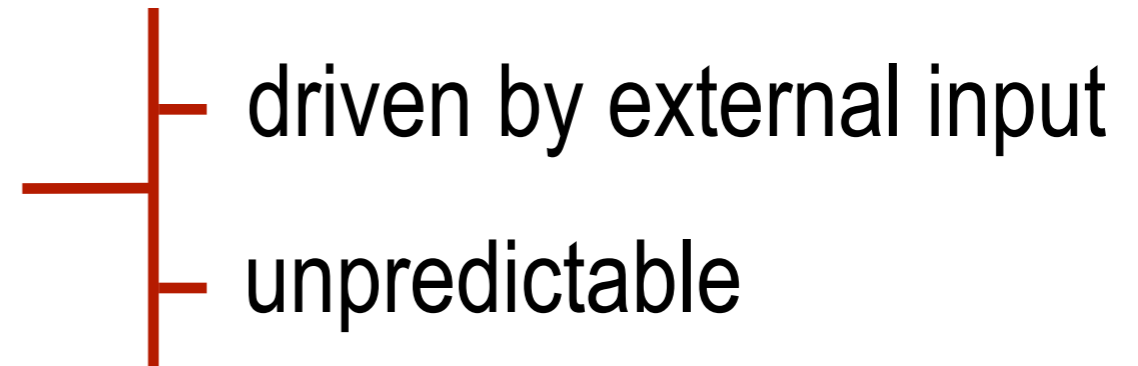
- **sense / discover** changes in computational surrounding (local or networked)
- dynamically adapt **behaviour** to change.

- Context:

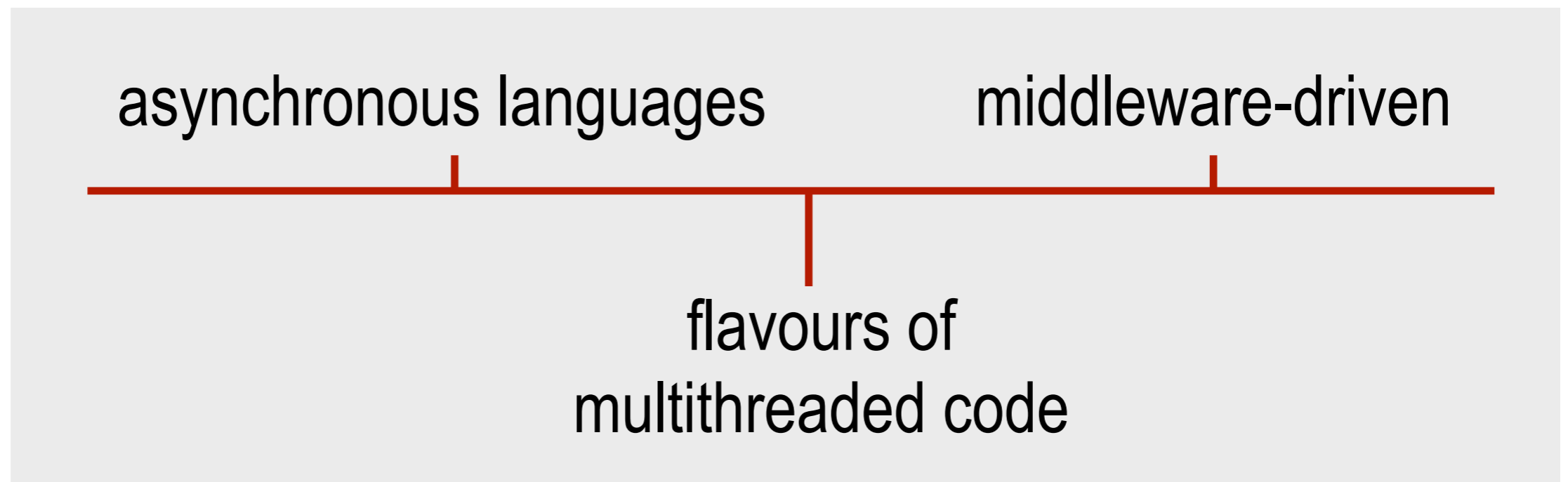
- any environmental information relevant to application behaviour; **unpredictable**

...or in fewer metaphors:

- Application's execution environment:



- Programming behaviour:



And effort goes towards...



Verifying application **behaviour**

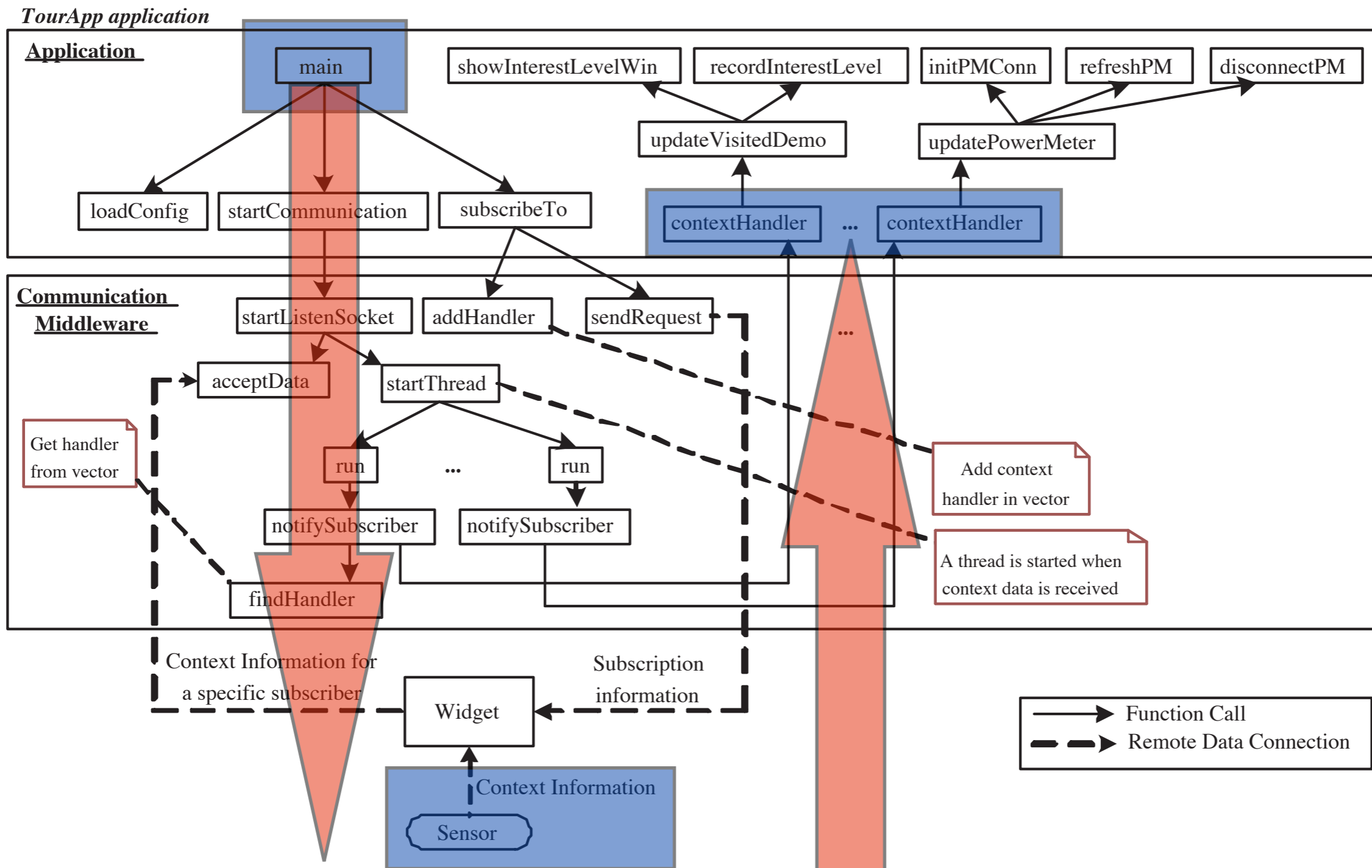


network-wide

at **node**

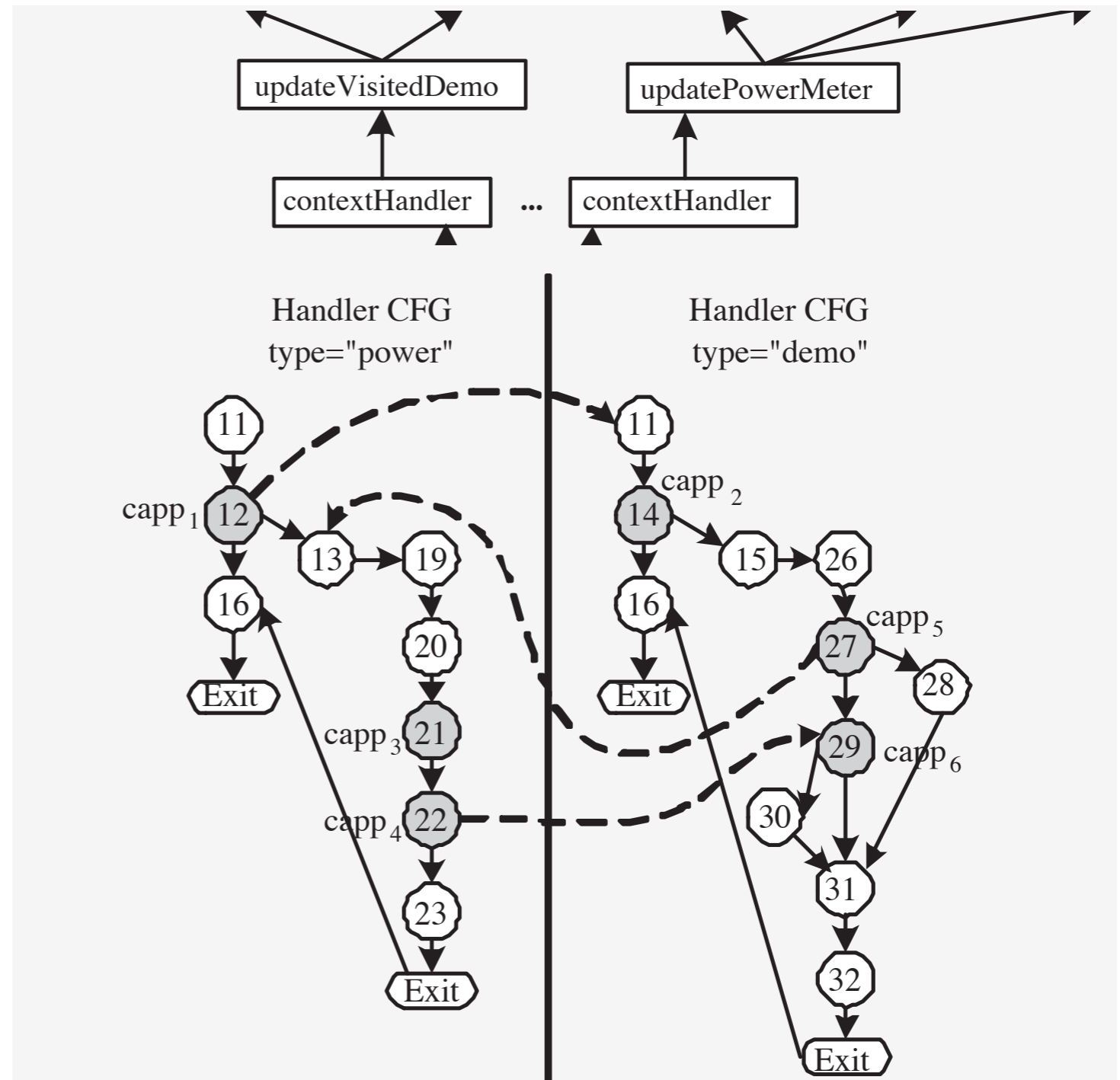
Related:

Keywords: middleware, Java, validation.



Related:

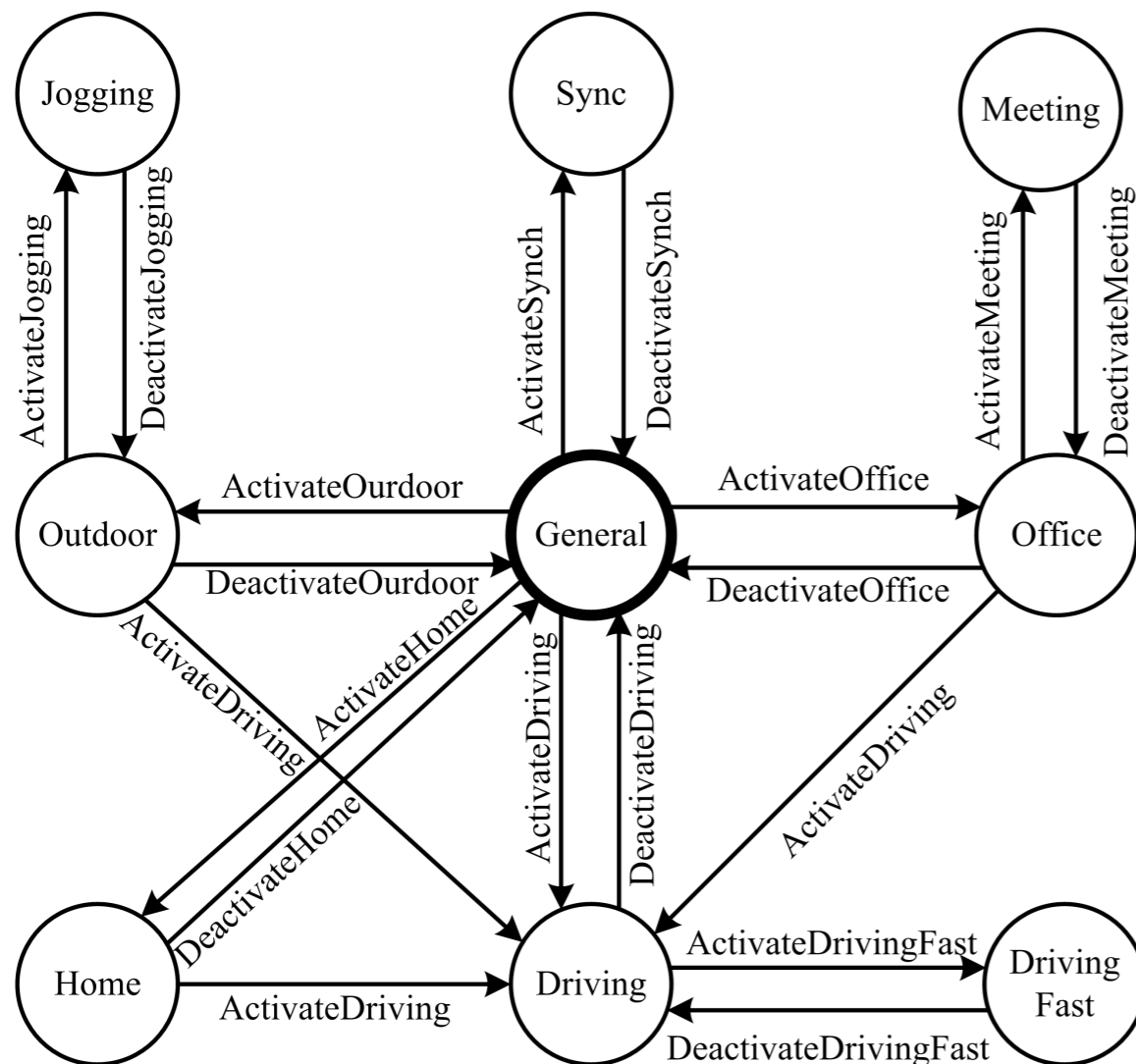
Keywords: middleware, Java, validation.



context-aware
program points (capps):
read/write

Related:

Keywords: rule-based adaptation model, symbolic verification



Faults:

- reachability
- determinism
- state/rule liveness
- stability

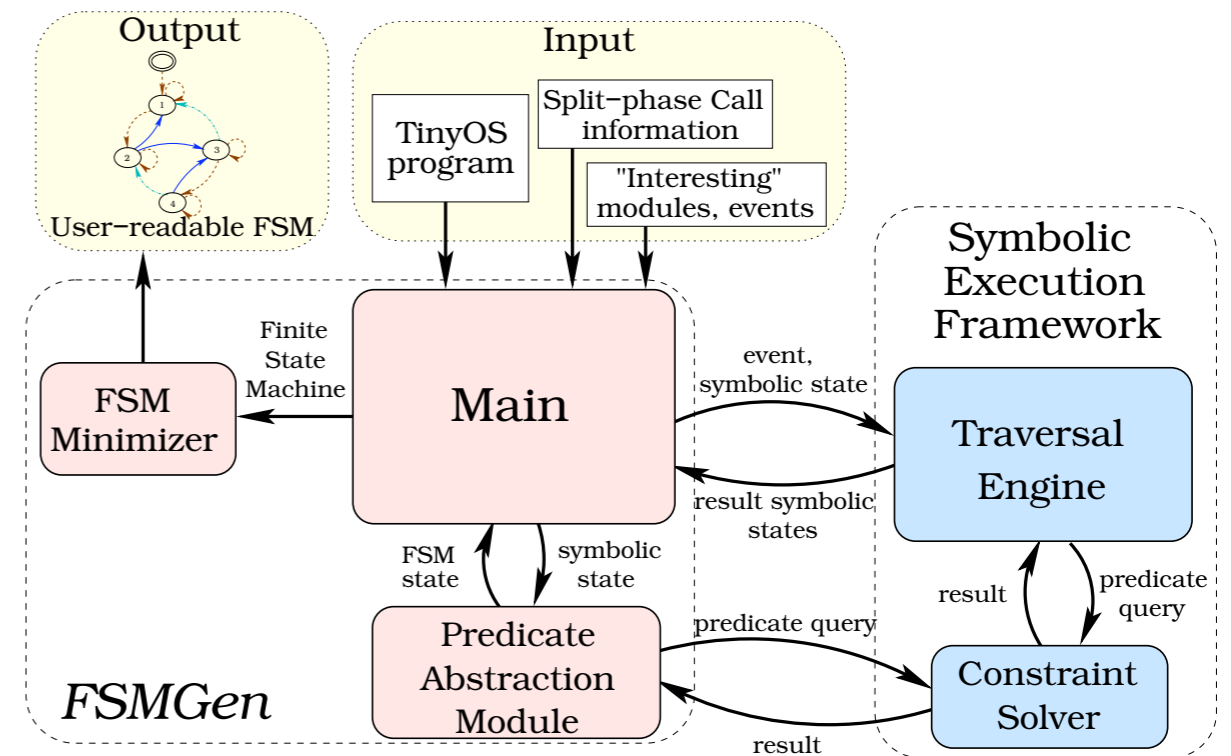
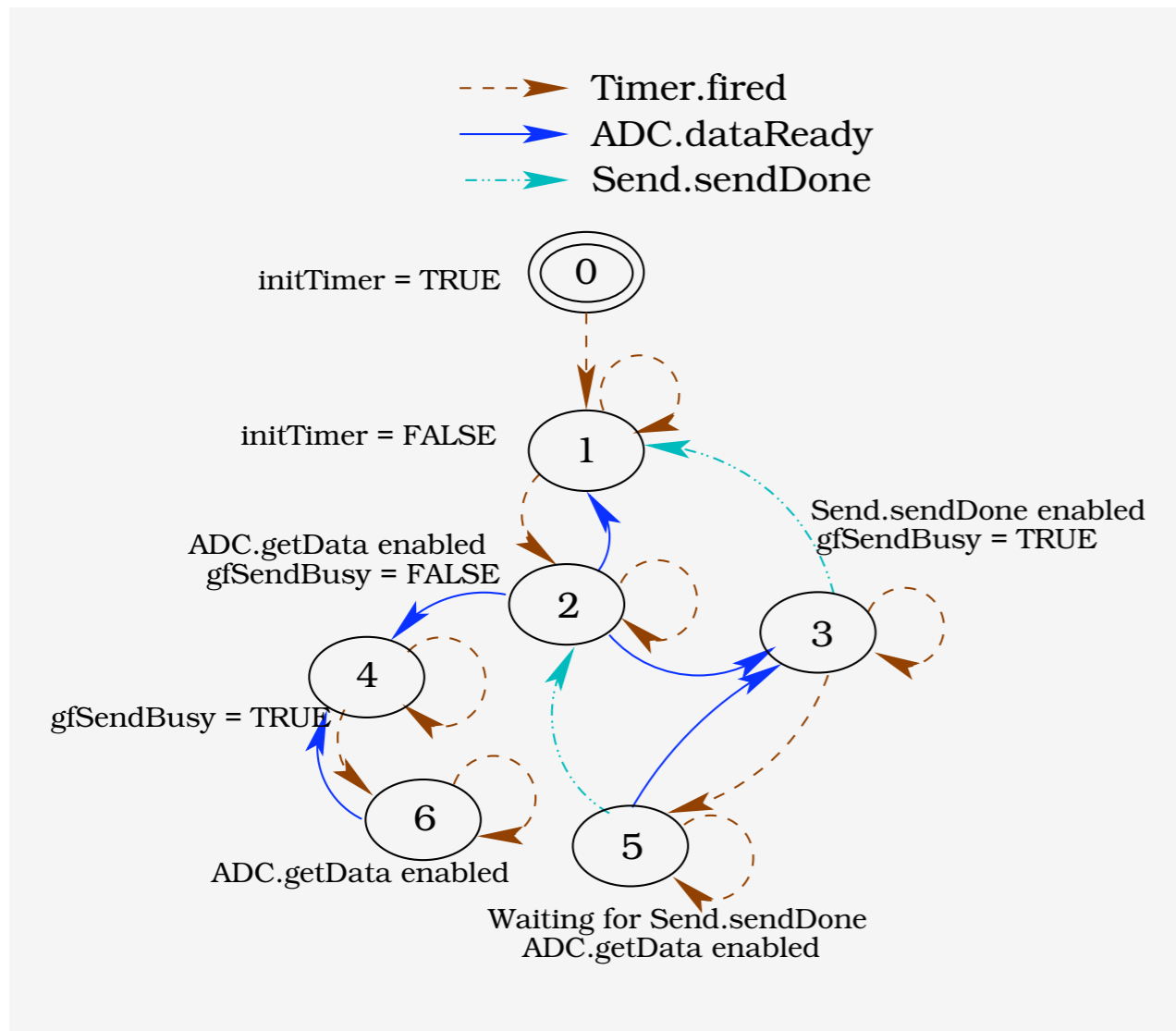
Related:

Keywords: high-level model, validation

Situation	$visitor_nearby = \langle C_{in}, p_{in}, computIllum \rangle$
Contexts	$C_{in} = \{d, V_{oc}, R, I, I_{max}, E_v\}$
Triggering condition	$p_{in} = (d \leq 5)$
Adaptive action	$computIllum \{$ $\$V_{oc} = V_{oc};$ $\$R = R;$ $I = \$V_{oc} / (r + \$R);$ $\$I = I$ $\$P = k * \$I * \$I * r;$ $\$d = d;$ $E_v = \$P / (\$d * \$d);$ $\}$

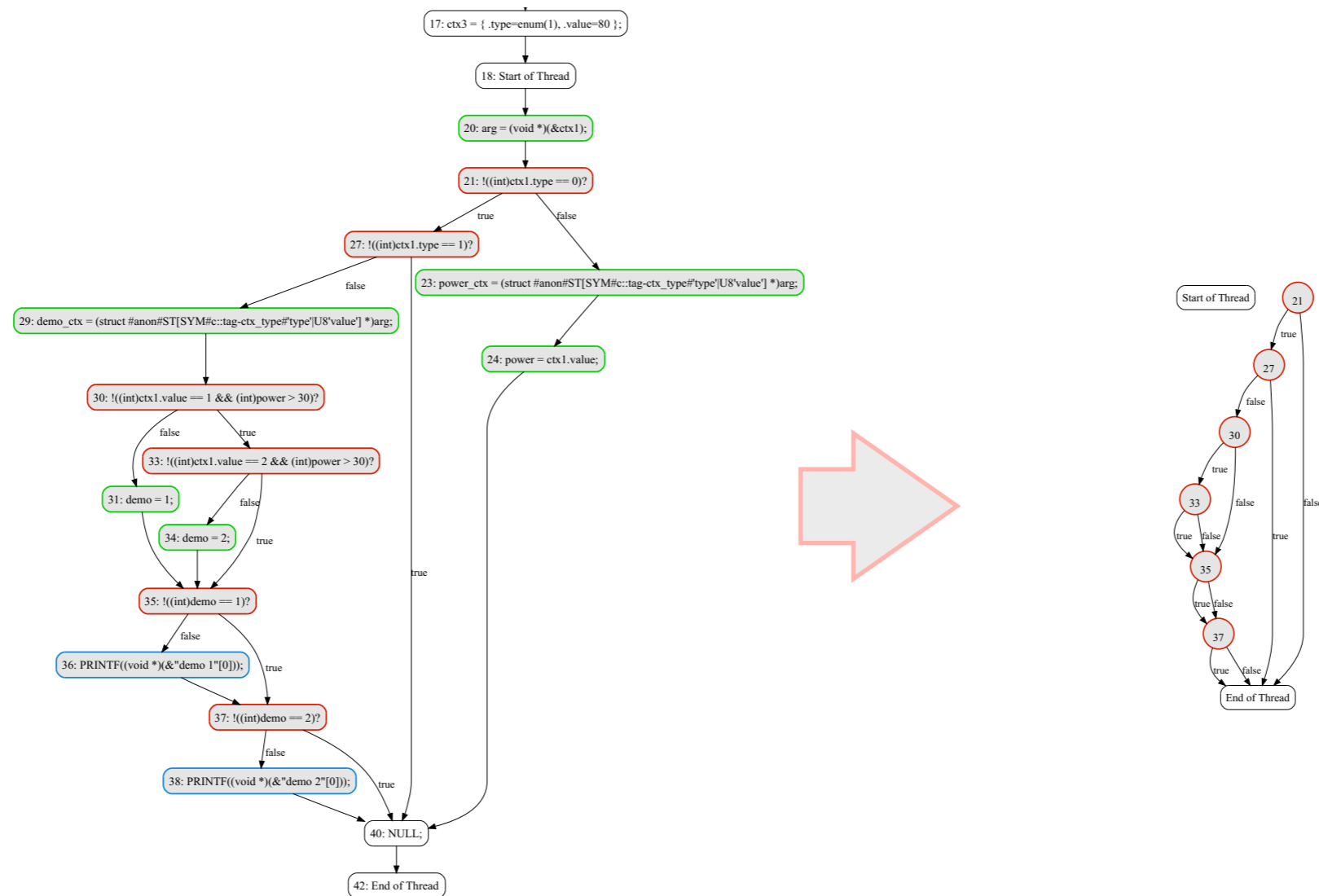
Related:

Keywords: asynchronous, nesC, model extraction.



These being said,

Model Extraction from Context-Aware Code



Capps: a generalization

- ... of side-effect-, escape-like analyses

- **read** capps

```
if (x > MAX)
    y = x;
```

- **write** capps

```
x = ctx;
```

- **termination** capps

```
if (x > MAX)
    while 1;
y = 1;
```

- **user** capps

```
print*(ctx);
or if (ctx)
    set_register(r);
```

... or in pictures:

```
unsigned power, demo;
```

```
void update_power(ctxt *power_ctx)
{
    power = power_ctx->value;
}
```

```
void update_demo(ctxt *demo_ctx)
{
    if(demo_ctx->value == 1 && power>30)
        demo = 1;
    else if(demo_ctx->value == 2 && power>30)
        demo = 2;

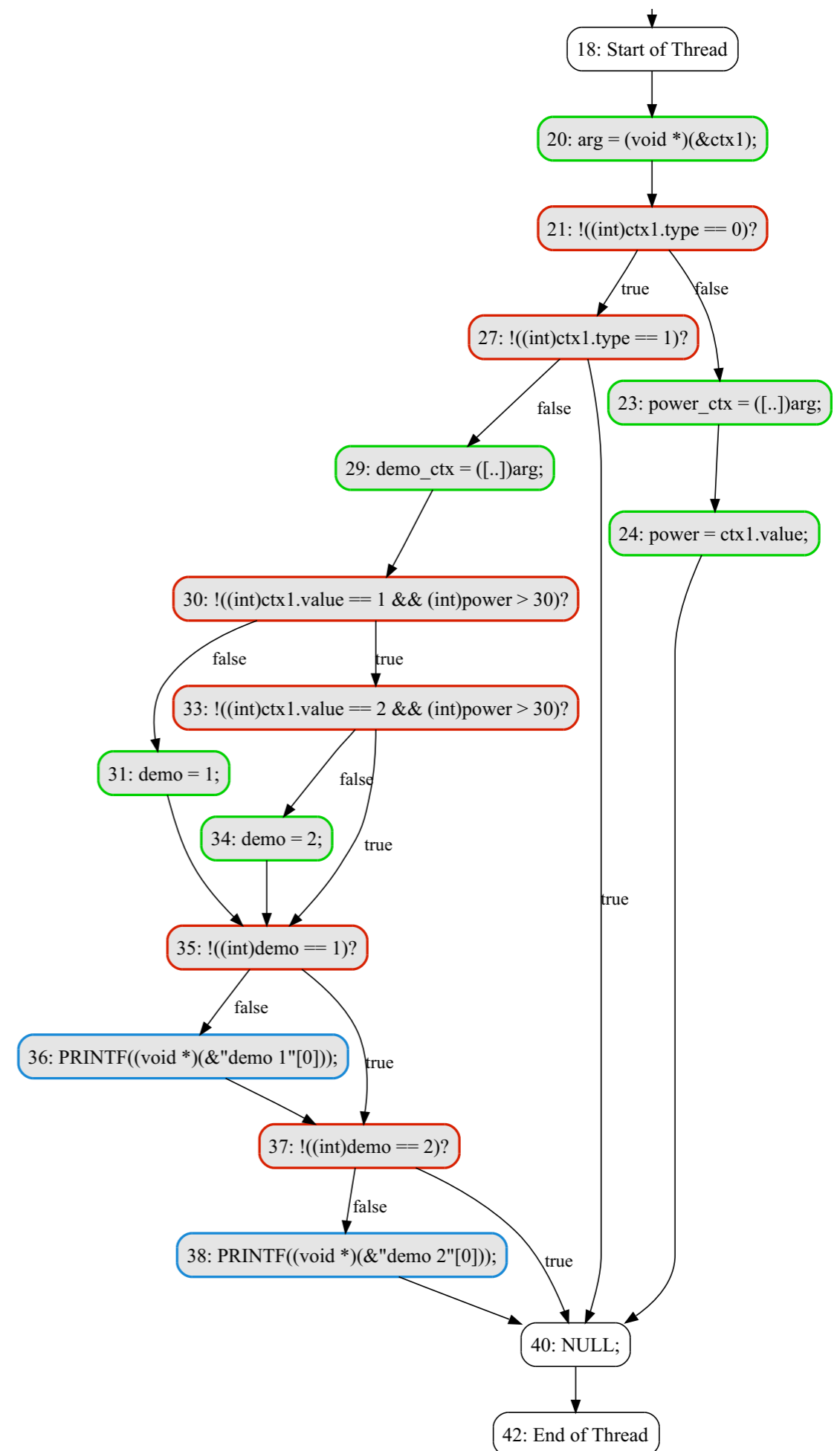
    if(demo == 1)
        printf("demo 1");
    if(demo == 2)
        printf("demo 2");
}
```

```
void* context_handler(void *arg)
{
    if(((ctxt *)arg)->type == TYPE_POWER)
        update_power(arg);

    else if(((ctxt *)arg)->type == TYPE_DEMO)
        update_demo(arg);

    return NULL;
}
```

```
void middleware()
{
    ctxt ctx1 = [...];
    pthread_create(id, NULL, context_handler, &ctx1);
}
```

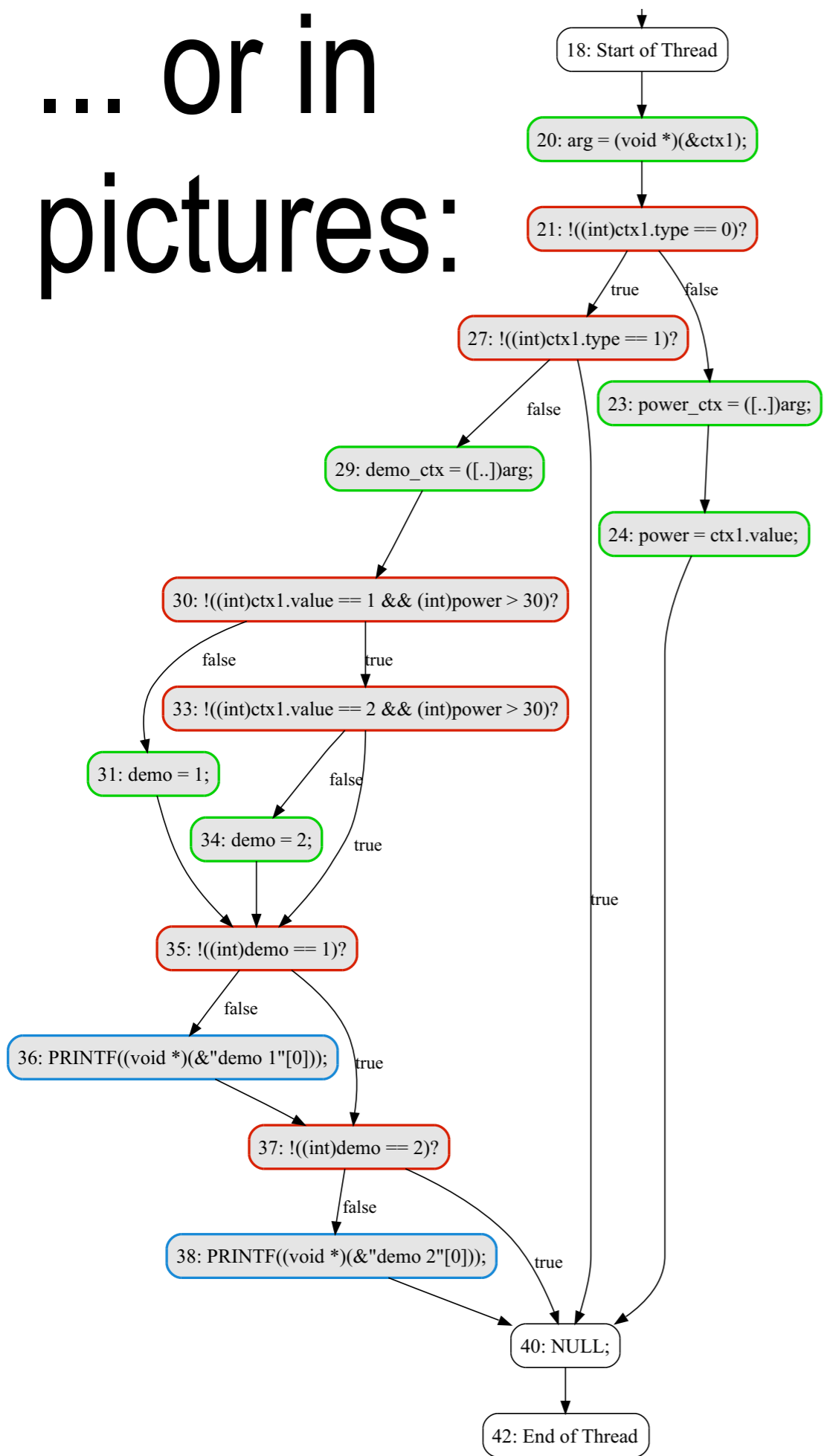


To FSM:

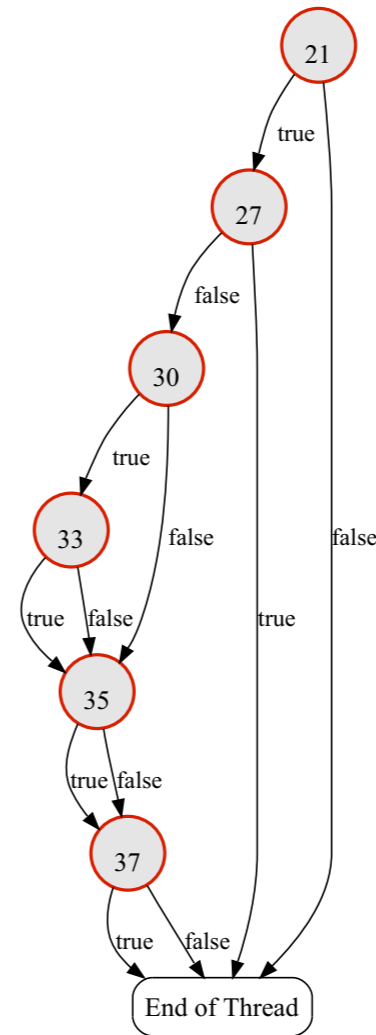
- FSM:
 - situations
<contexts, predicate, action>
 - ... to state machine:
transitions, states
- Take e.g.
 - transitions: all context-controlled control flow
 - states: all other side effects

Situation <i>visitor_nearby</i> = $\langle C_{in}, p_{in}, computIllum \rangle$	
Contexts	$C_{in} = \{d, V_{oc}, R, I, I_{max}, E_v\}$
Triggering condition	$p_{in} = (d \leq 5)$
Adaptive action	$computIllum \{$ $\$V_{oc} = V_{oc};$ $\$R = R;$ $I = \$V_{oc} / (r + \$R);$ $\$I = I$ $\$P = k * \$I * \$I * r;$ $\$d = d;$ $E_v = \$P / (\$d * \$d);$ $\}$

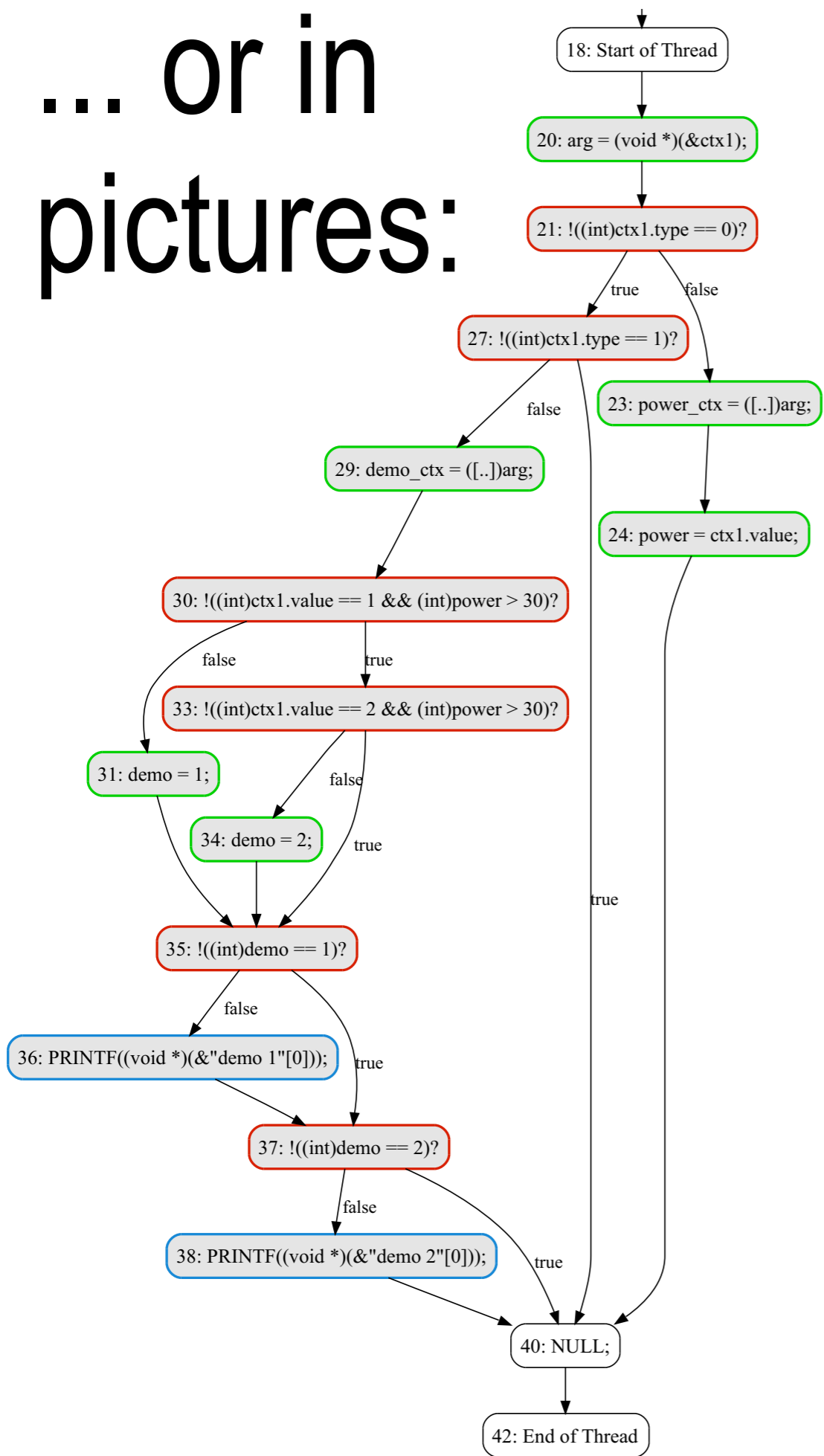
... or in pictures:



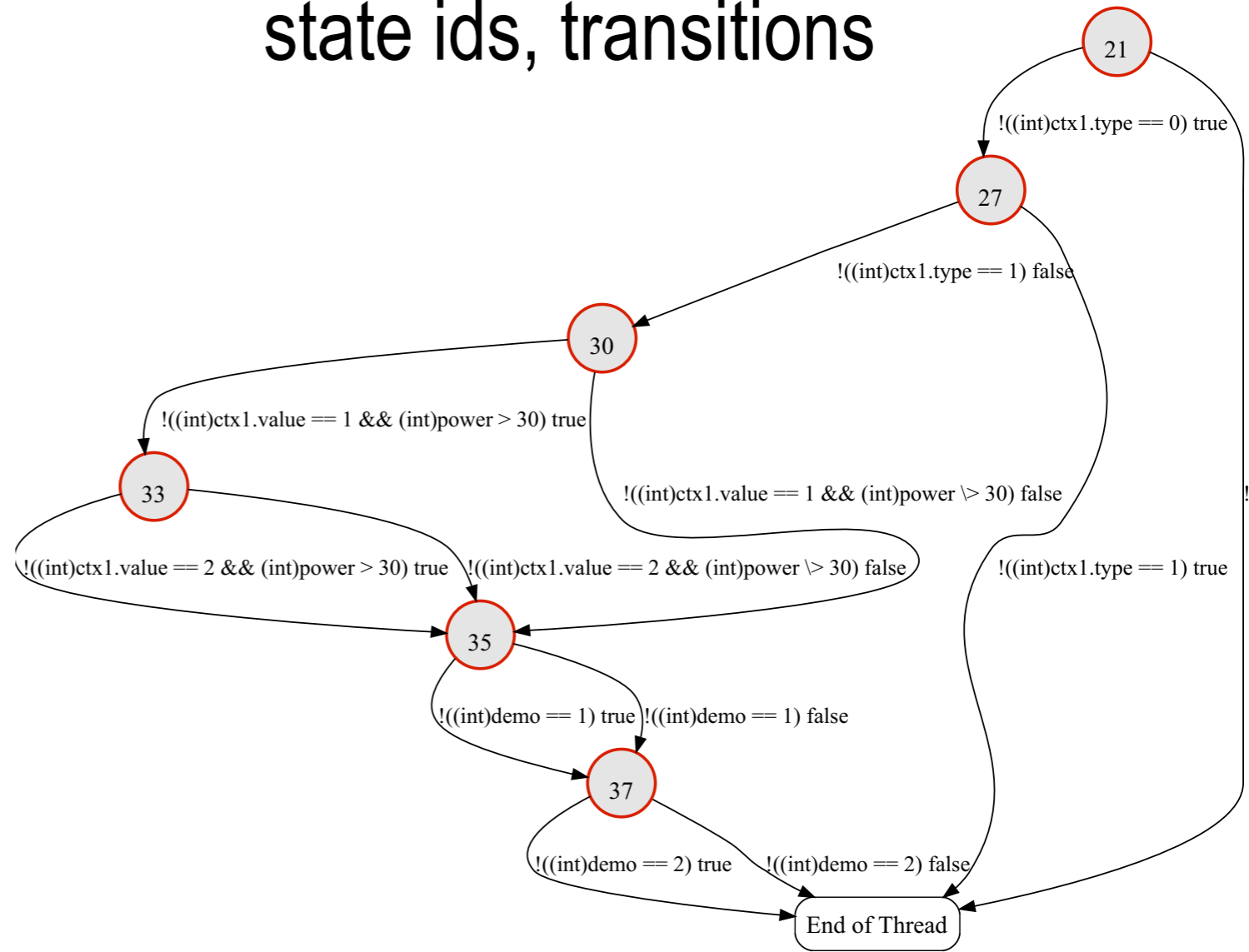
state ids



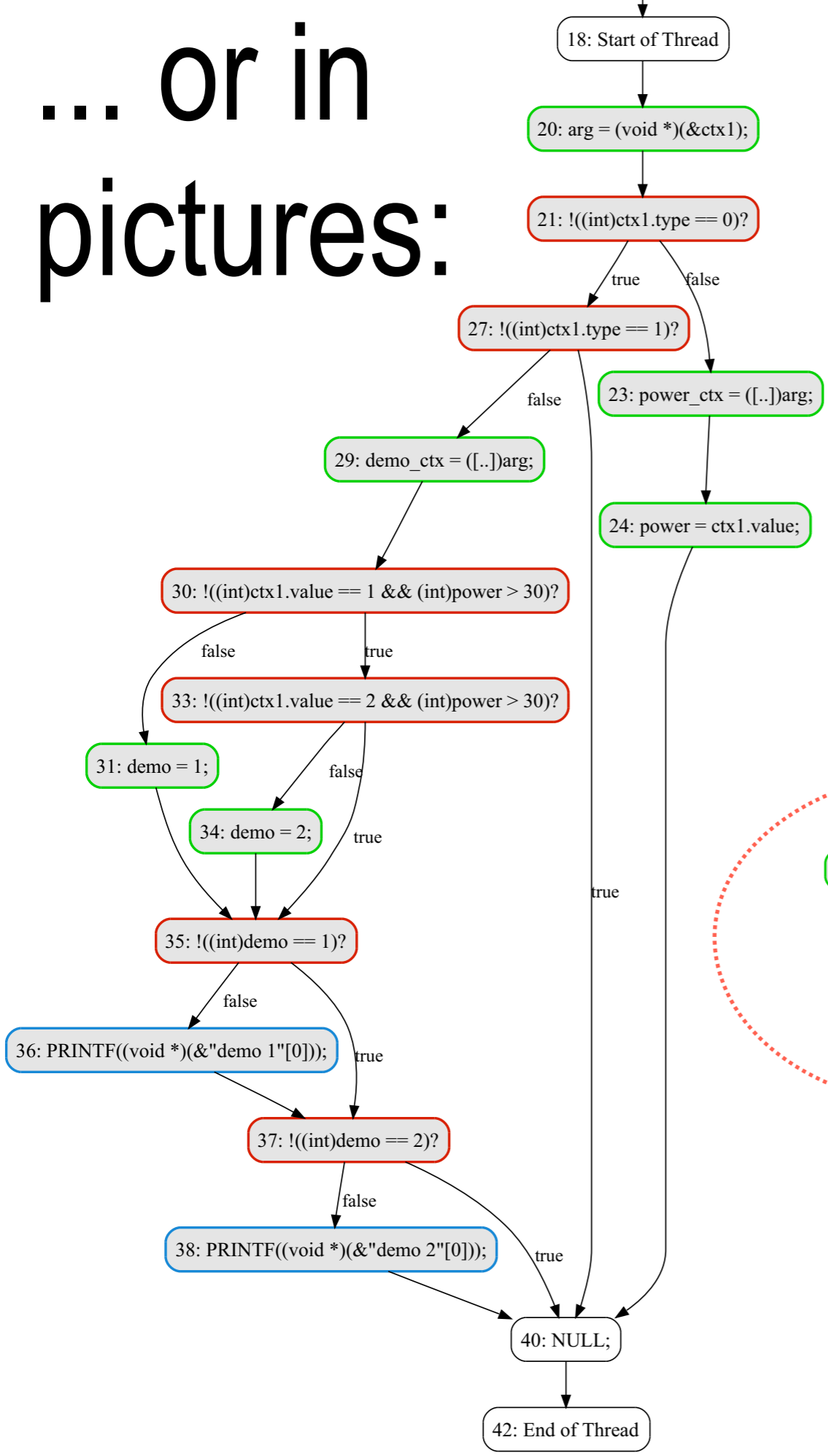
... or in pictures:



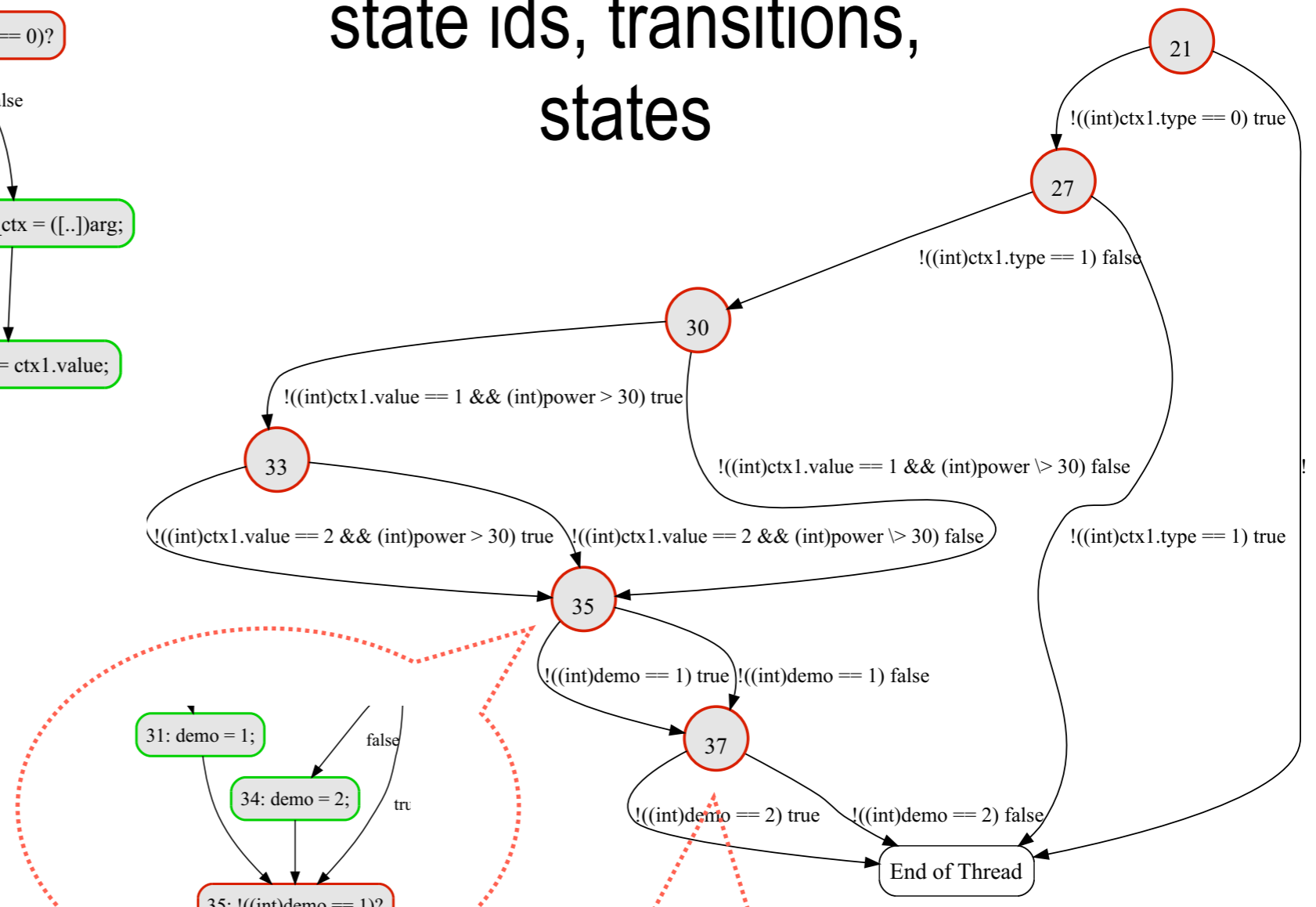
state ids, transitions



... or in pictures:



state ids, transitions, states



```

// state composition
(18, 20, 21)
(23, 24, 40, 42)
(27)
(29, 30)
(31, 35)
(33)
(34, 35)
(35)
(36, 37)
(37)
(38, 40, 42)
(40, 42)
  
```