Using Instrumentation for Quality Assessment of Resilient Software in Embedded Systems

David Lawrence[1] Didier Buchs[1] Armin Wellig[2]

[1] University of Geneva

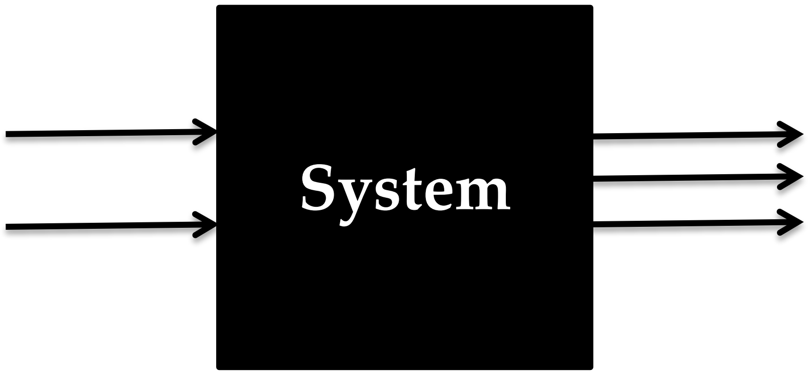
[2] Honeywell International Sarl

16/10/2014

6th International Workshop on Software Engineering for Resilient Systems

Introduction

- System correctness



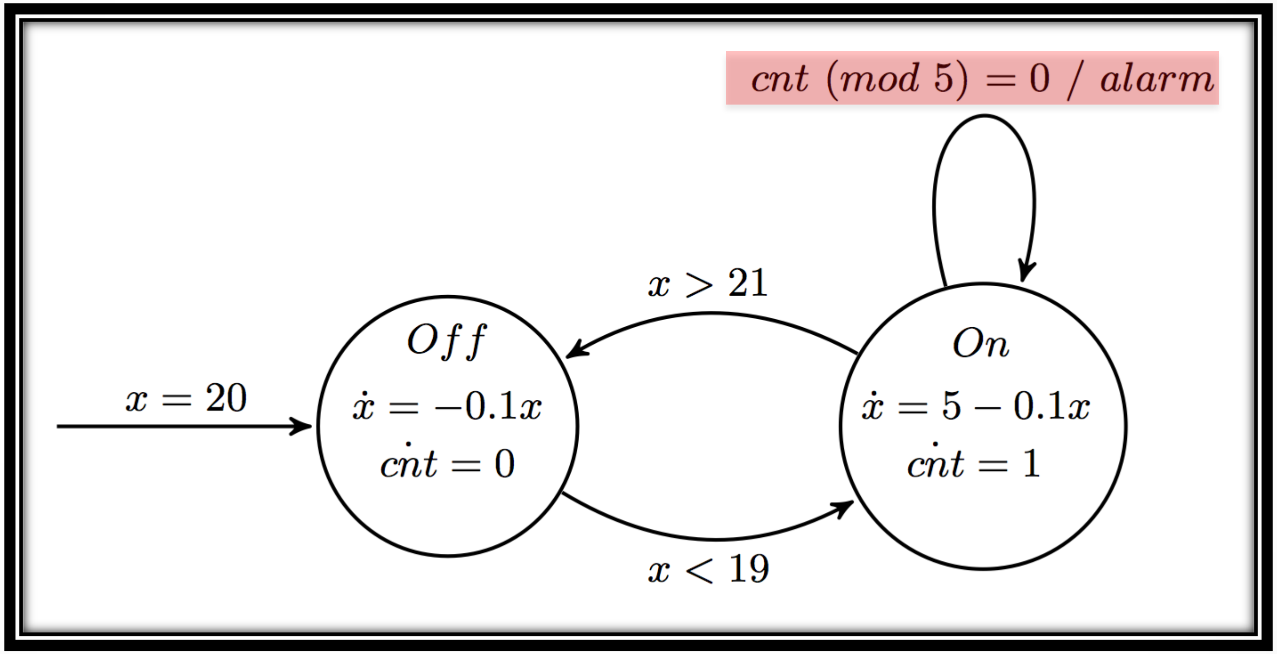
- Correct internal state **OR** false positive?
- What about resilient system correctness?
- Ideas/Solutions => **Improve observability**

System observation

- Automaton specification
  - T(spec) are words ∈ $(\Sigma_{in} \times \Sigma_{out})^*$
- Satisfaction relation
  - For any program *p* based on a given specification *spec*

$$\boxed{p \vDash spec \Leftrightarrow T(p) \equiv T(spec)}$$

Limited observability



- What if the developer made a fault with the counter modulo?

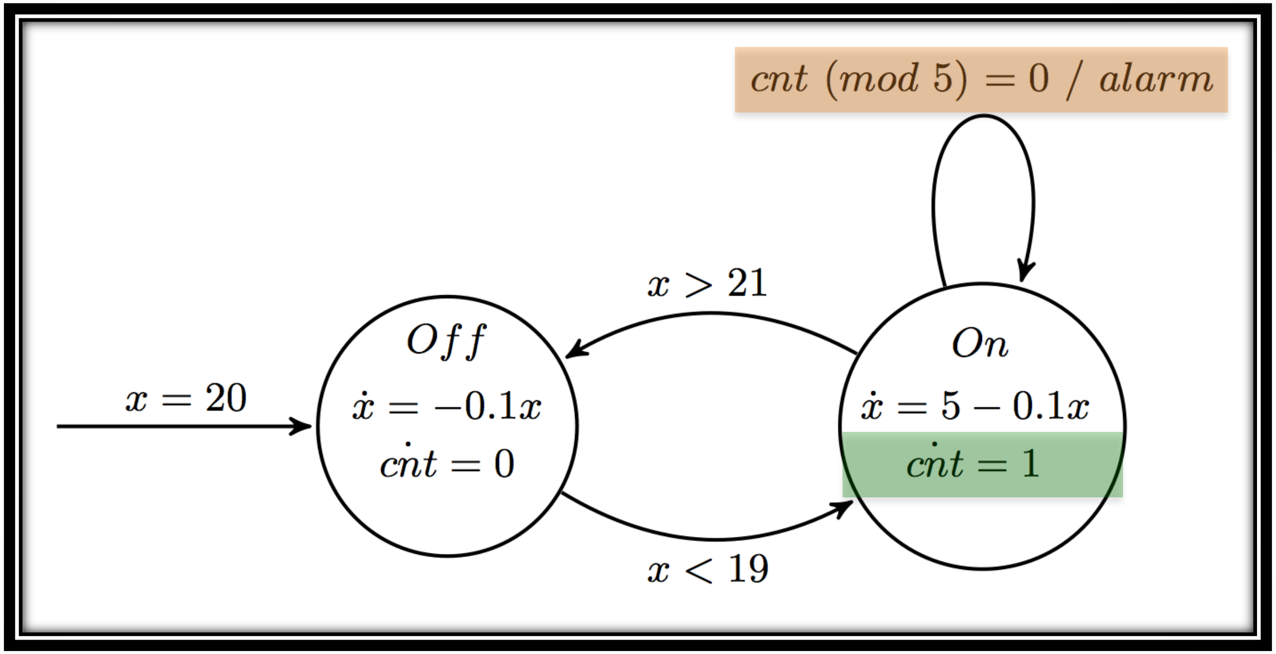
Improve observability

- Adding new observation points *obs* to a program *p*

- For any program *p*, *obs* are correct observers iff

$$\boxed{\Pi_{spec}\big(T(p + obs)\big) = T(p)}$$

Observability: example



- **Obs1**: each increment of *cnt*
- **Obs2**: when "*cnt mod 5 = 0*"

Observers expressiveness

- For a given program *p*, the observers *obs* expressiveness can *be defined as follow*

$$\boxed{expr(p, obs) = \sum_{t1 \in T(p+obs)} |t1|}$$

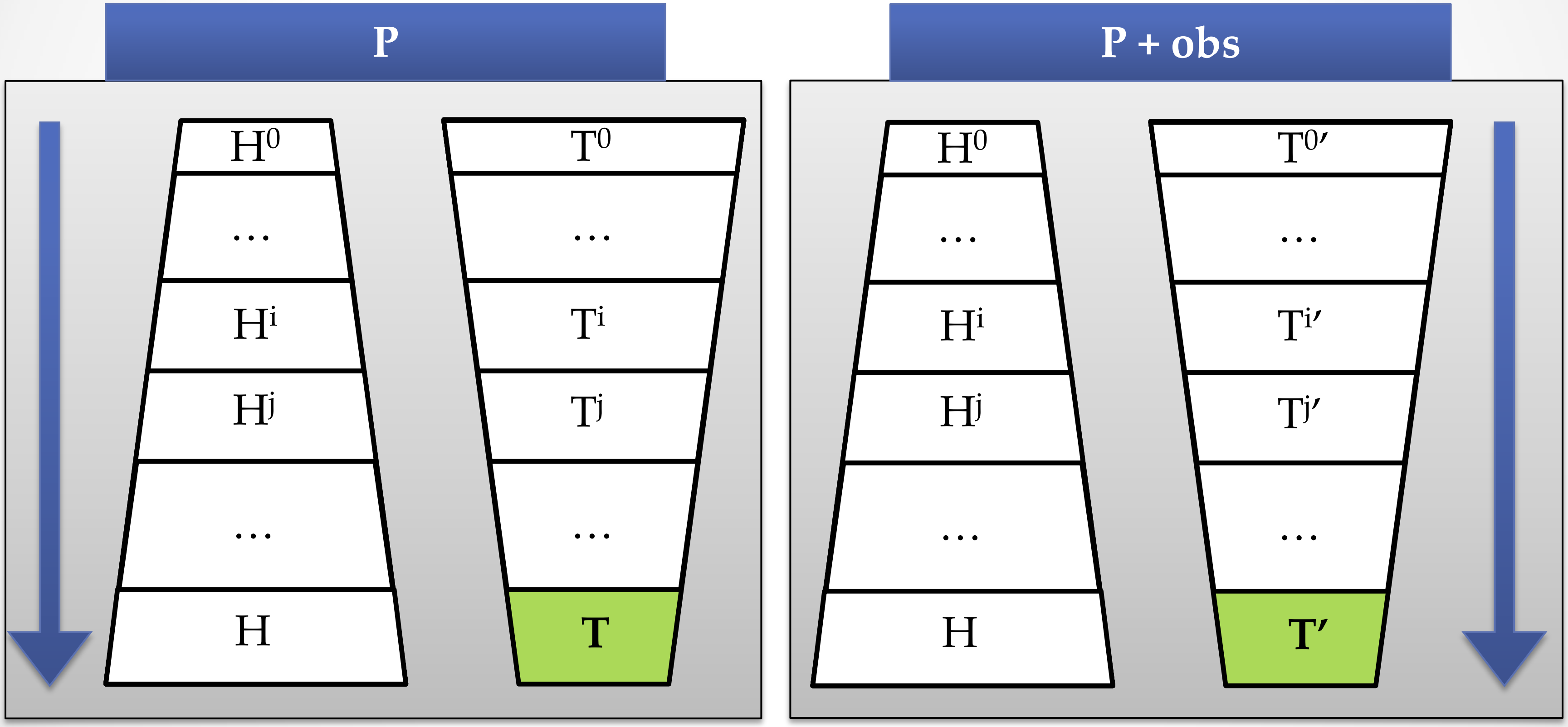
Metric: distance

- A distance between instrumented program traces and specification traces can be stated

$$\boxed{\left| expr(p, obs) - \sum_{t2 \in T(spec)} |t2| \right|}$$

- What about the observers relevance?

Metric: quality/relevance



P

H^0 … H^i H^j … H

T^0 … T^i T^j … **T**

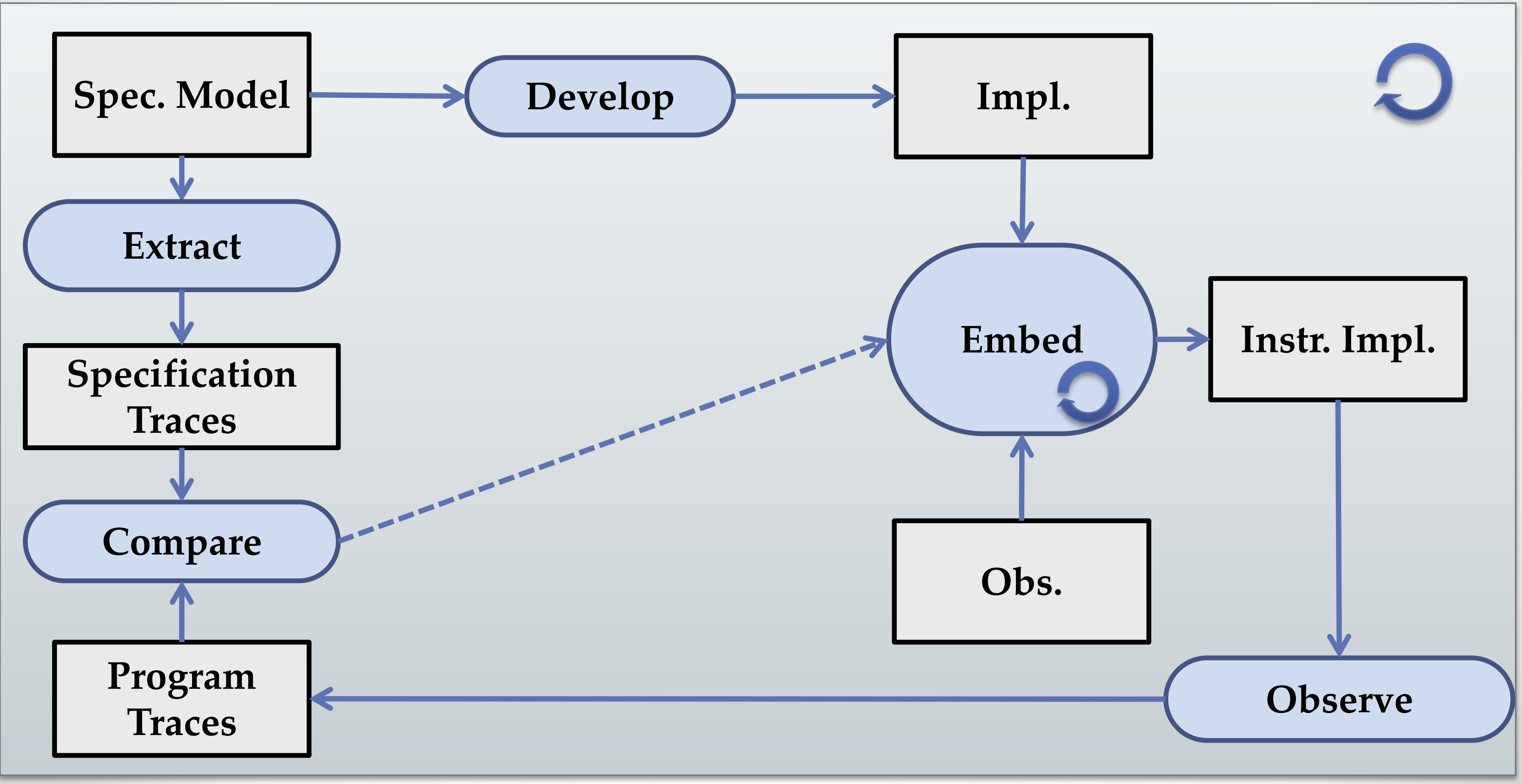
P + obs

H^0 … H^i H^j … H

$T^{0\prime}$ … $T^{i\prime}$ $T^{j\prime}$ … **T′**

T >> T’

Instrumentation process



Instrumentation: example

```
void Thermostat (int t) {
    if (t > 21) {
        t += -0.1 * t;                  // Increase temperature
    } else if (t < 19) {
        t += 5 – 0.1 * t;               // Decrease temperature
    }
    return t;
}
```

t = 21 → t=18.9 → t=21.29 → t=19.161

Instrumentation: example

```
void Thermostat (int t) {
    if (t > 21) {
        t += -0.1 * t;                    // Increase temperature
    } else if_instr (t < 19) {
        t += 5 – 0.1 * t;                 // Decrease temperature
    }
    return t;
}
```

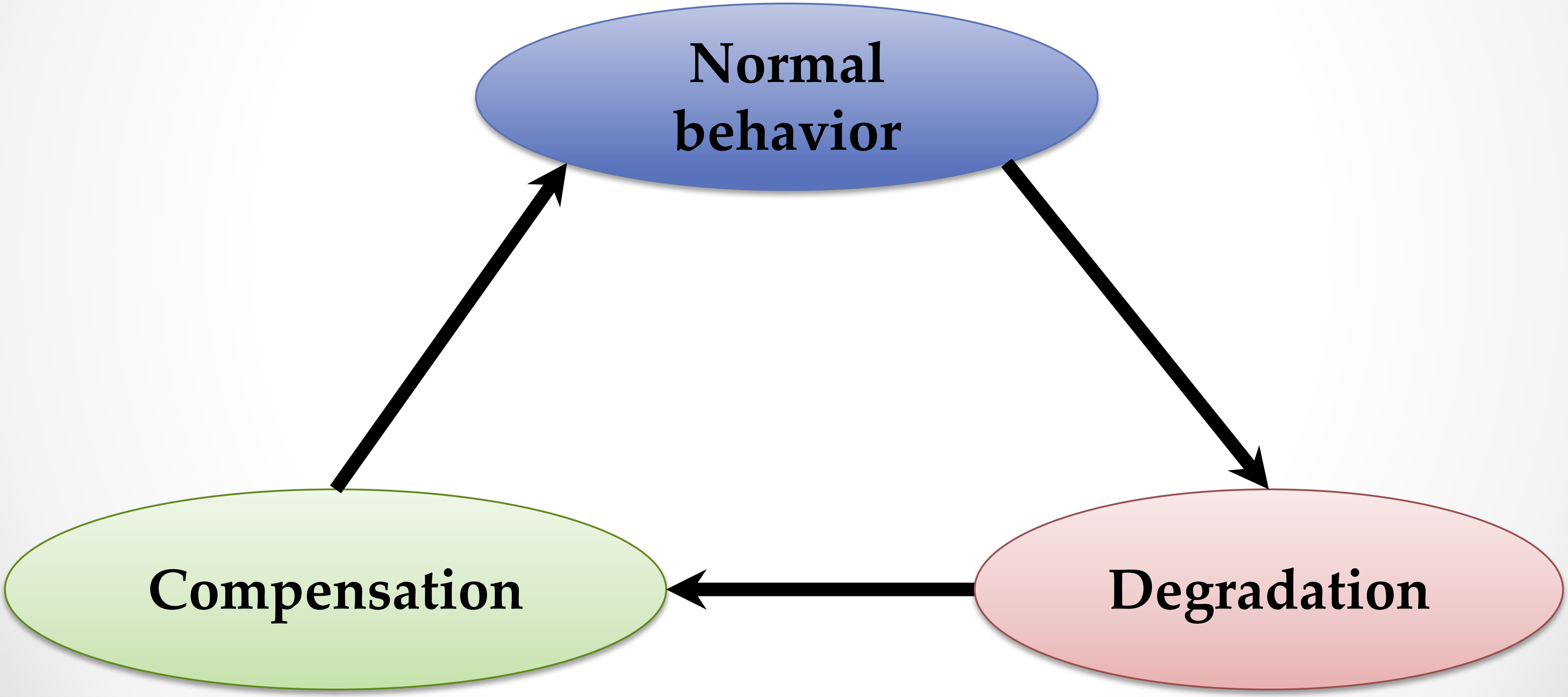
t = 21 → t=18.9 → **t < 19**→ t=21.29 → t=19.161

Instrumentation: example

```
void Thermostat (int t) {
    if_instr (t > 21) {
        t += -0.1 * t;                    // Increase temperature
    } else if_instr (t < 19) {
        t += 5 – 0.1 * t;                 // Decrease temperature
    }
    return t;
}
```

t = 21 → t=18.9 → **t < 19**→ t=21.29 → **t > 21** → t=19.161

Resilient system



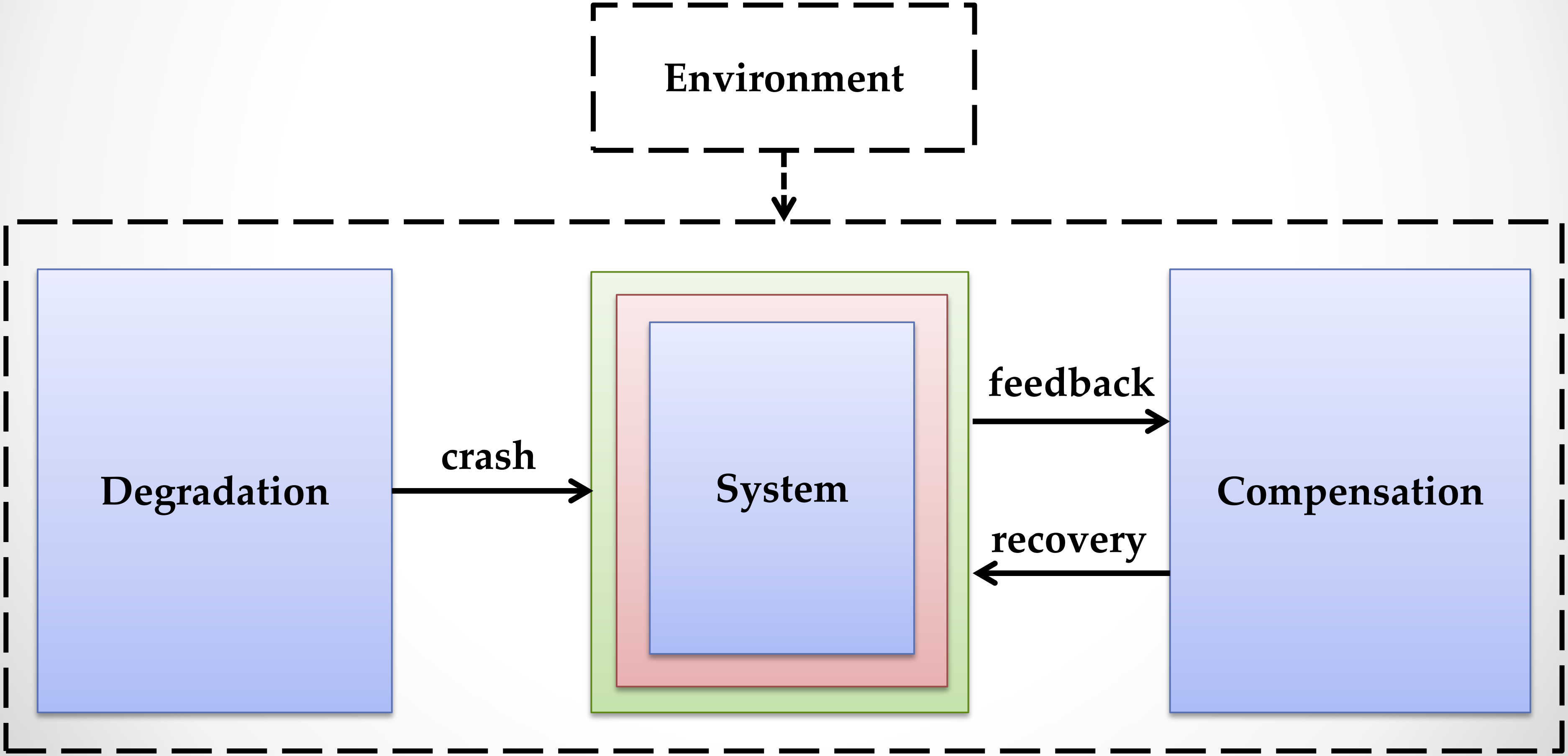
Instrumenting resilient systems

- Resilient system designed with defined FSMs
- System crashes considered as degradation
- Instrumentation requirements
- LTL used to discuss properties

Instrumenting resilient systems

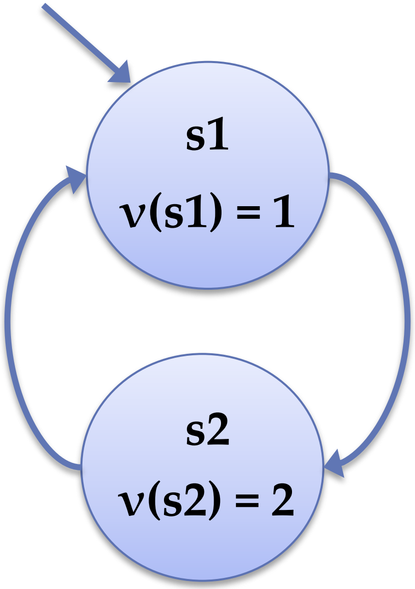
- Extended system
  - $F_{sys}'' = F_{sys} + feedback + recovery + obs_{sys}$
- Compensation
  - $F_{comp}' = F_{comp} + obs_{comp}$
- Degradation
  - $F_{deg}' = F_{deg} + obs_{deg}$

High level picture



Preparing the system

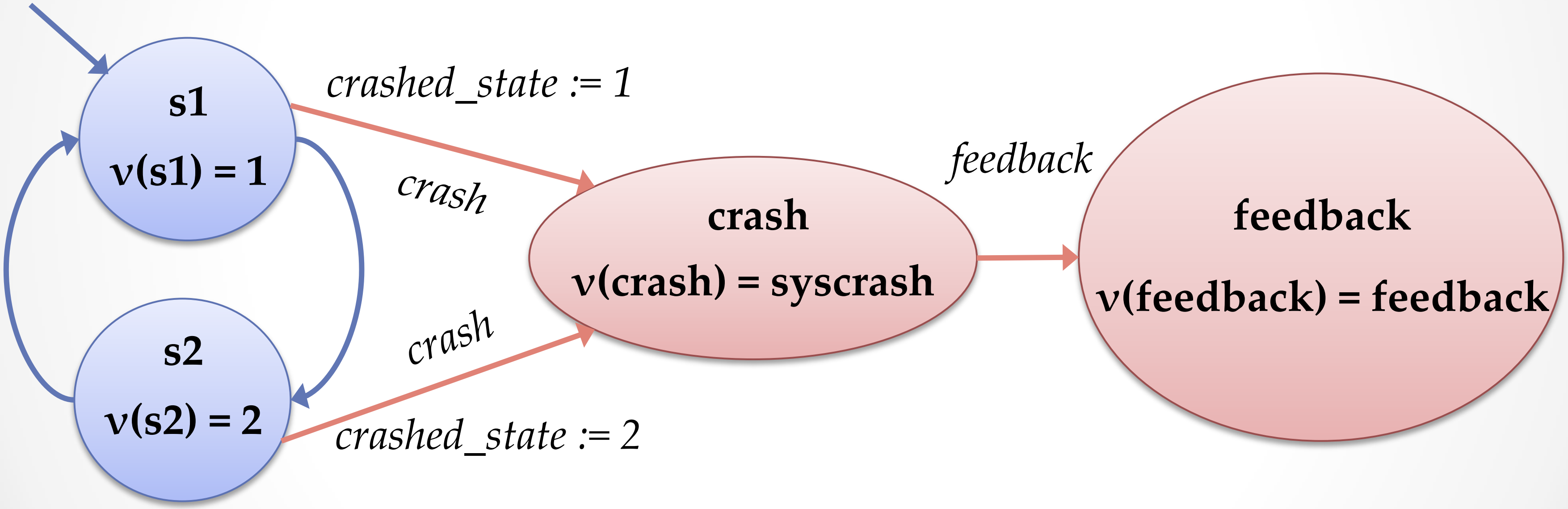
- Preliminary preparation
  - Add atomic propositions to distinguish states



Extending the system

- 1st extension F_{sys}' (Feedback)
  1) Add crash state
  2) Add feedback state
  3) Add arcs from every state to crash state
     - *crash* as arc inscription
     - Unique state ID stored in *crashed_state* variable
  4) Add arc between crash and feedback
     - *feedback* as arc inscription
  5) Add atomic propositions

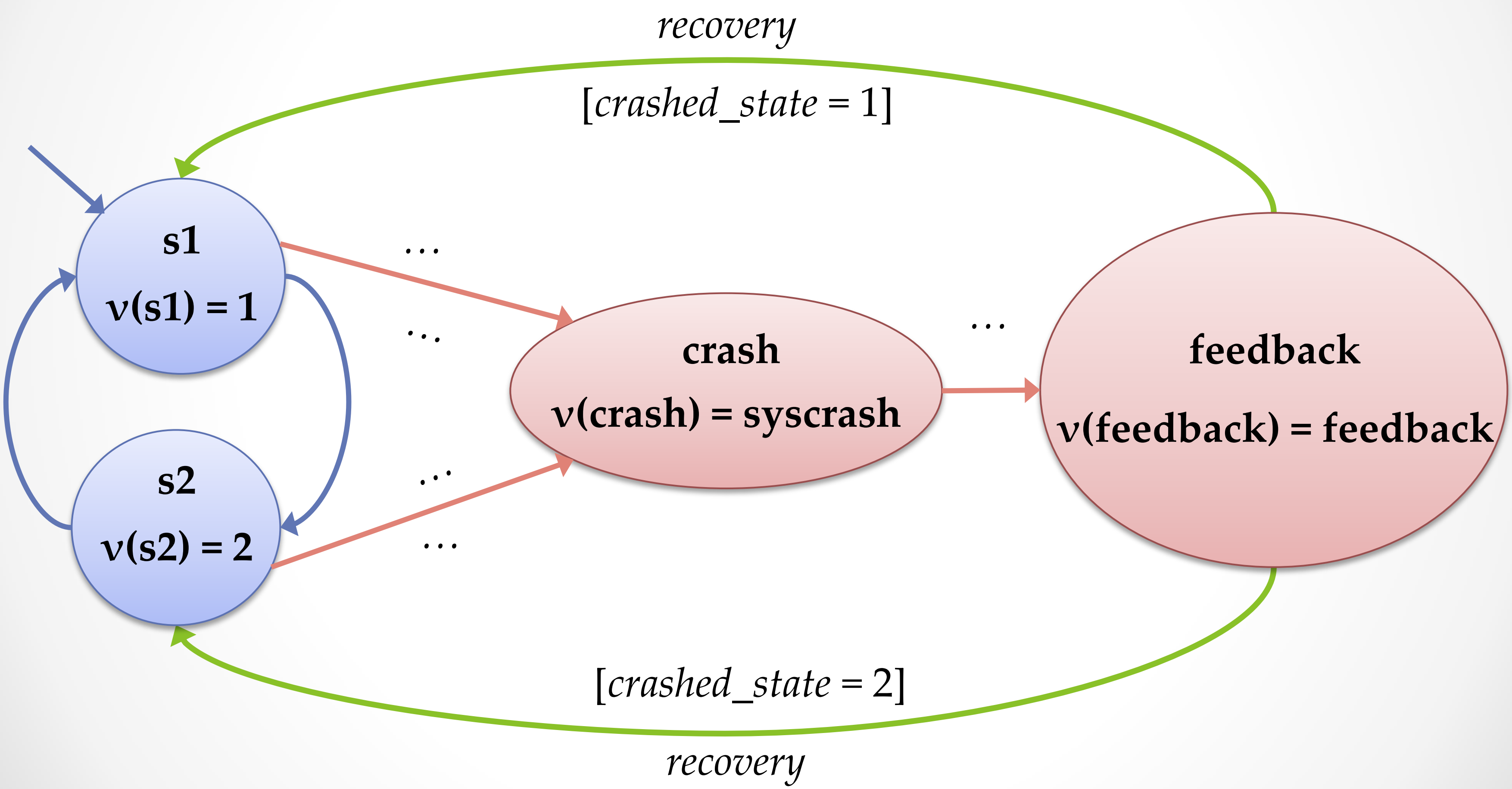
Extending the system



Extending the system

- 2nd extension F_{sys}'' (Recovery handling)
  1. Add arcs from feedback state to every system state
     - *recovery* as arc inscription
  2. Guards
     - Avoid non determinism
     - Recovery to last consistent state using *crashed_state*

Extending the system



recovery

[*crashed_state* = 1]

s1

ν(s1) = 1

s2

ν(s2) = 2

…

…

…

…

crash

ν(crash) = syscrash

…

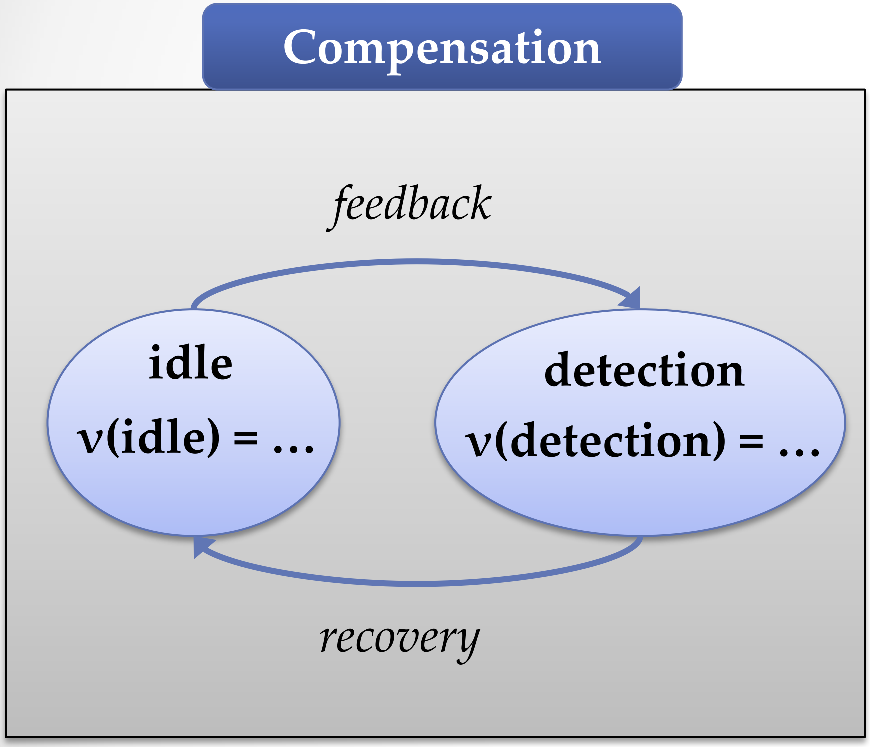
feedback

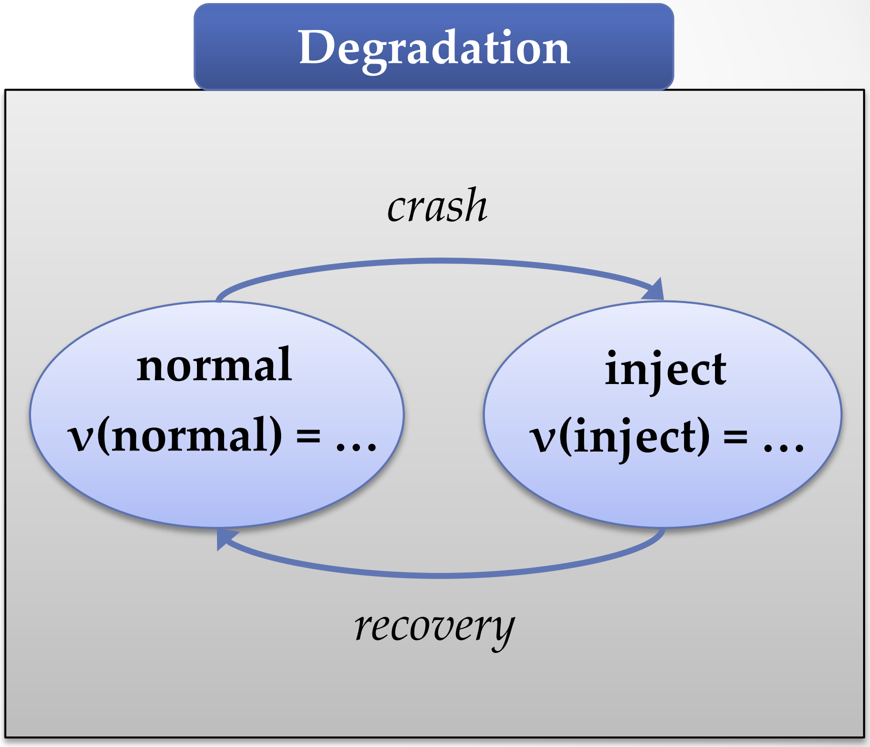
ν(feedback) = feedback

[*crashed_state* = 2]

recovery

Compensation and degradation





System composition

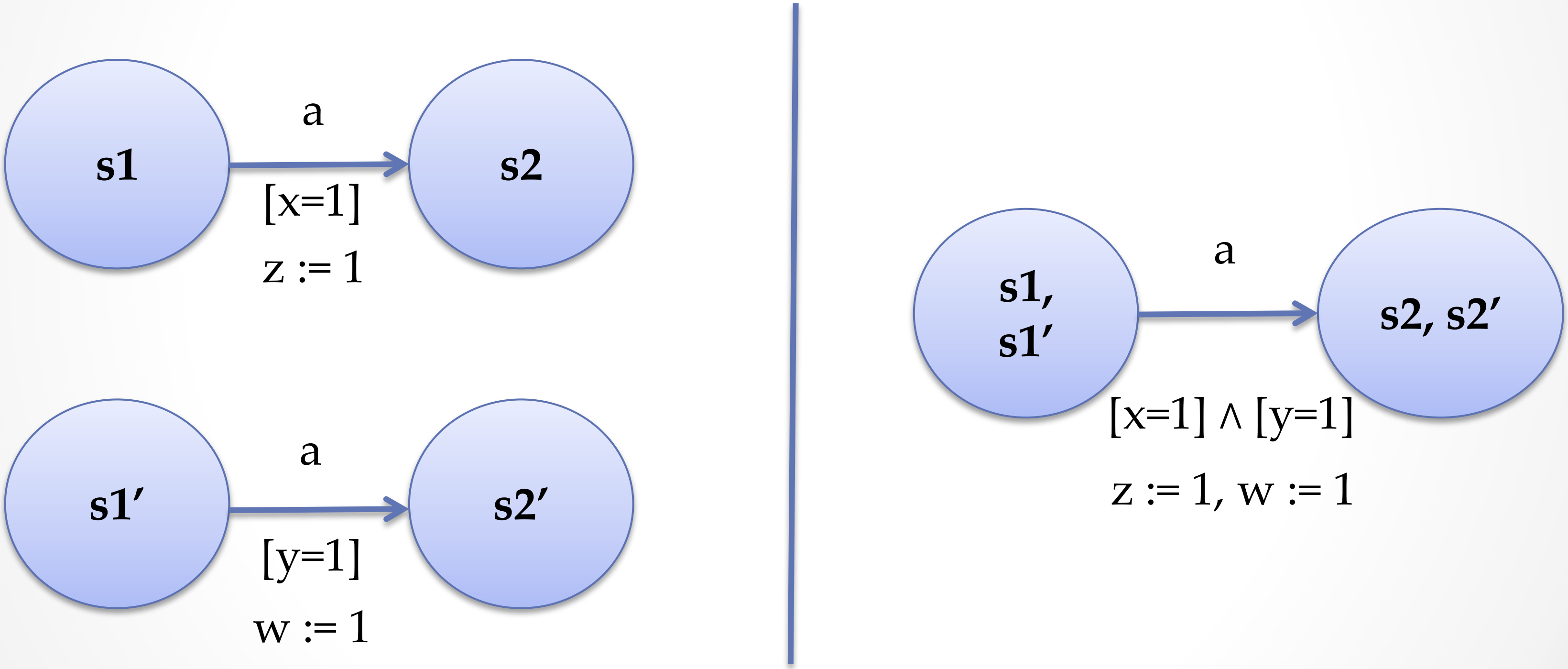
- Complete resilient system
  - $F_{res} = F_{sys}'' \mid\mid F_{comp}' \mid\mid F_{deg}'$

- Composition
  1. Add arc looping on each state with εas inscription
  2. Synchronized product of the two FSM

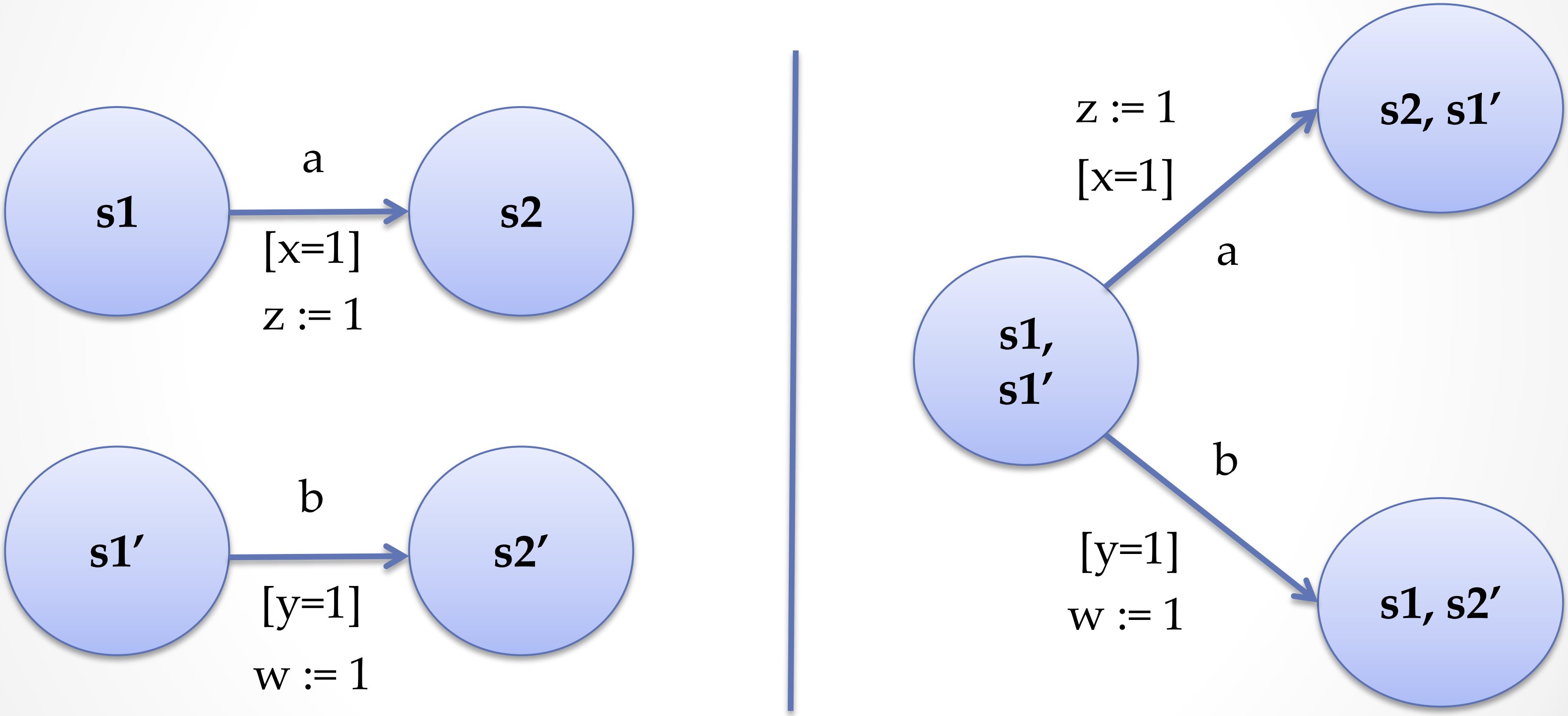
System composition

- $FSM_{compo} = FSM_1 \ || \ FSM_2$
  - Union of the alphabets
  - Cartesian product of states
  - Cartesian product of initial states
  - Union of variables
  - Union of atomic propositions

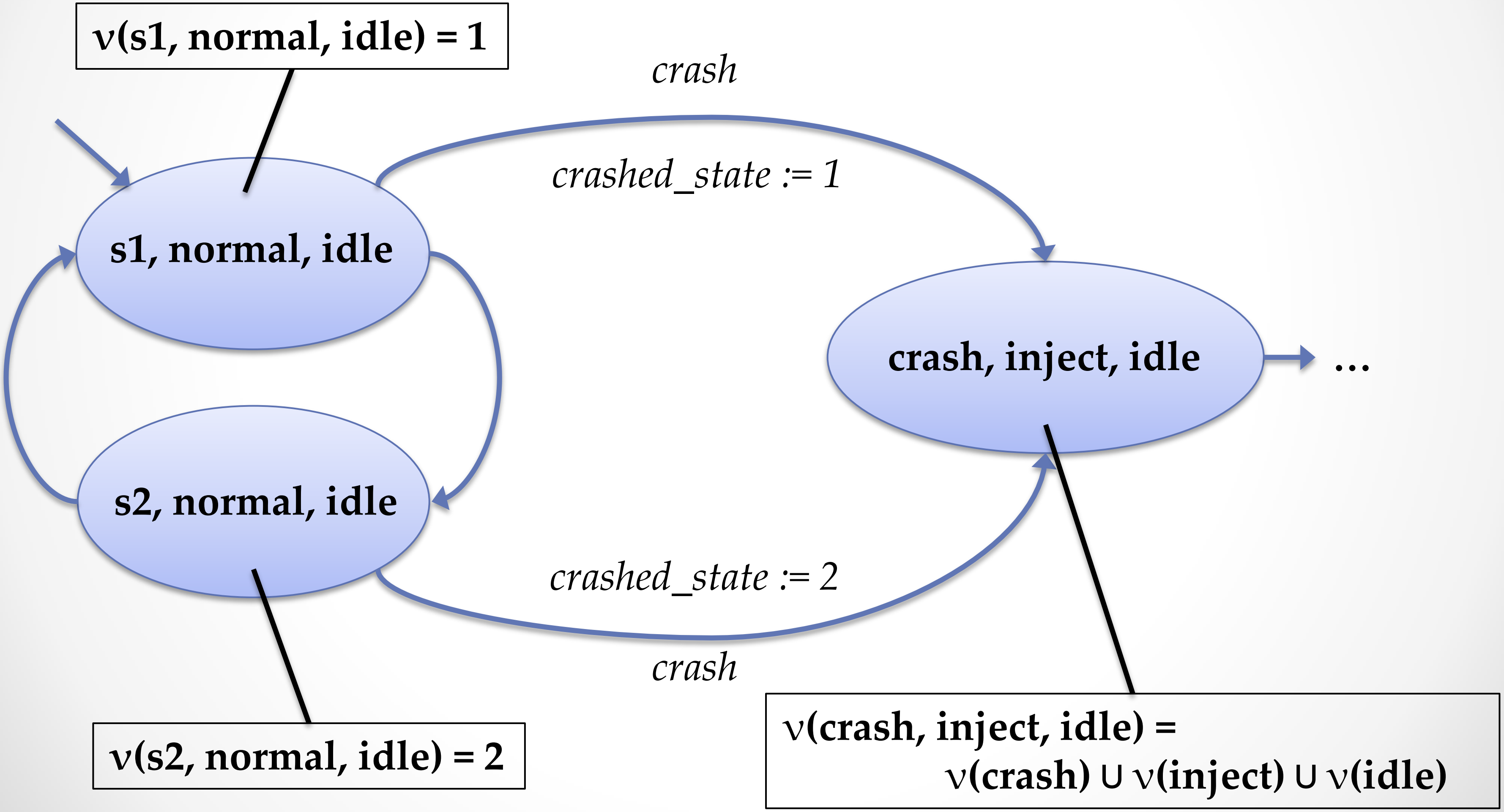
System composition



System composition



System composition



ν(s1, normal, idle) = 1

crash

crashed_state := 1

s1, normal, idle

crash, inject, idle

…

s2, normal, idle

crashed_state := 2

crash

ν(s2, normal, idle) = 2

ν(crash, inject, idle) =
ν(crash) ∪ ν(inject) ∪ ν(idle)

Resiliency properties

- System's resiliency

$$\boxed{G\left(\left((1 \vee 2 \vee 3) \wedge X\, syscrash\right) \rightarrow X\left(F(1 \vee 2 \vee 3)\right)\right)}$$

- Improvements

$$\boxed{\begin{gathered} G(((1 \wedge X\, syscrash) \rightarrow X\left(F(1)\right)) \vee \\ ((2 \wedge X\, syscrash) \rightarrow X\left(F(2)\right)) \vee \\ ((3 \wedge X\, syscrash) \rightarrow X\left(F(3)\right))) \end{gathered}}$$

Model checking

- StrataGEM [López et al. 2014]
  - Symbolic model-checker
  - Using concepts of Term Rewriting
  - Using Decision Diagrams for data representation

Model checking

- Usage
  - Resilient system translated as a transition system
  - Strategies/rewriting rules defined independently for each components
  - State space computed

Conclusion

- Theoretical basis on instrumentation
- Insights on resilient systems instrumentation
- Methodology to extend a system with resilience
  - Even though the model is simple
- Temporal properties enunciated
  - Mechanisms and overall resiliency

Future works

- Level resiliency
  - More complete/complex model [Trivedi et al. 2009]

- Model checking
  - LTL with *StrataGEM* when available
  - Other model checkers

- Tests generation
  - Model based tests generation [Fraser et al. 2009]
  - Timing insights [Braberman et al. 1997]

Thank you

Questions ?

References

- Edmundo López Bóbeda, Maximilien Colange, and Didier Buchs. Stratagem: A generic petri net verification framework. In Gianfranco Ciardo and Ekkart Kindler, editors, *Petri Nets*, volume 8489 of *Lecture Notes in Computer Science*, pages 364– 373. Springer, 2014.
- Kishor S. Trivedi, Dong Seong Kim, and Rahul Ghosh. 2009. Resilience in computer systems and networks. In *Proceedings of the 2009 International Conference on Computer-Aided Design* (ICCAD '09). ACM, New York, NY, USA, 74-77.
- Fraser, G., Wotawa, F., & Ammann, P. E. (2009). Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, *19*(3), 215-261.
- Victor Braberman, Miguel Felder, and Martina Marr. Testing timing behavior of real-time software. In *International Software Quality Week, 1997*.