ONERA
THE FRENCH AEROSPACE LAB

La Région Occitanie Pyrénées · Méditerranée

LIRMM

SkiNet
A Petri Net Generation Tool for the Verification of Skillset-based Autonomous Systems

Baptiste Pelletier, Charles Lesire, David Doose, Karen Godary-Dejean, Charles Dramé-Maigné
Workshop on Formal Methods for Autonomous Systems

26/09/2022

Presentation plan

1. Context
2. Skillset
3. Translation
4. Verification
5. Conclusion
6. Background

Skill-based architecture?

This work verifies skill-based specifications of robots, as introduced by Albore et al. [1].

Skill-based software architecture:

Pros

- Simpler specification
- Easier verification
- Ideal for automation
- Doesn't need a high experience in robotics
- Controller code is generated from the model and follows the same semantic

Cons

- Requires a solid middleware
- Hardware issues are either overlooked or unpredictable
- Scalability issues for verification with increasing complexity

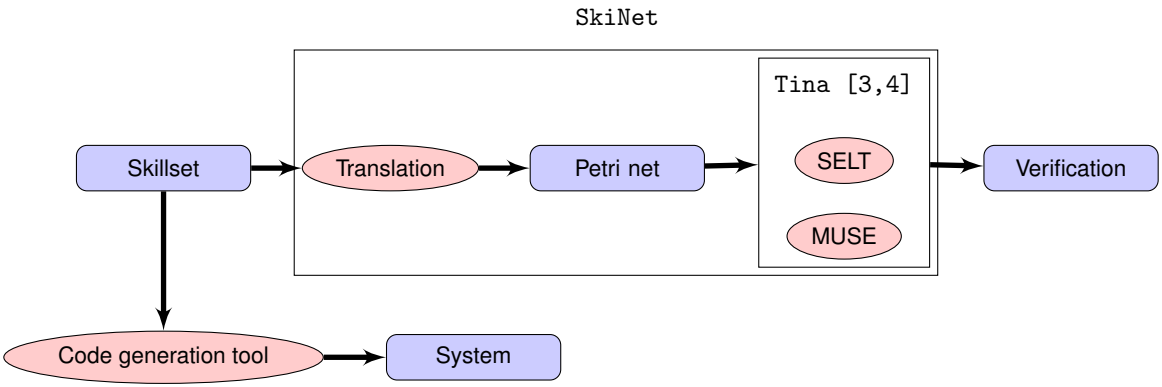
[1] Alexandre Albore, David Doose, Christophe Grand, Charles Lesire and Augustin Manecy (2021): Skill-Based Architecture Development for Online Mission Reconfiguration and Failure Management. In: 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), IEEE, pp. 47–54, doi: 10.1109/RoSE52553.2021.00015.

How much do you trust your robot?

Objective of SkiNet: improve the **trustability** of autonomous systems designed with **skill-based architectures** (Albore et al. [1]). Using **a-priori Verification**:

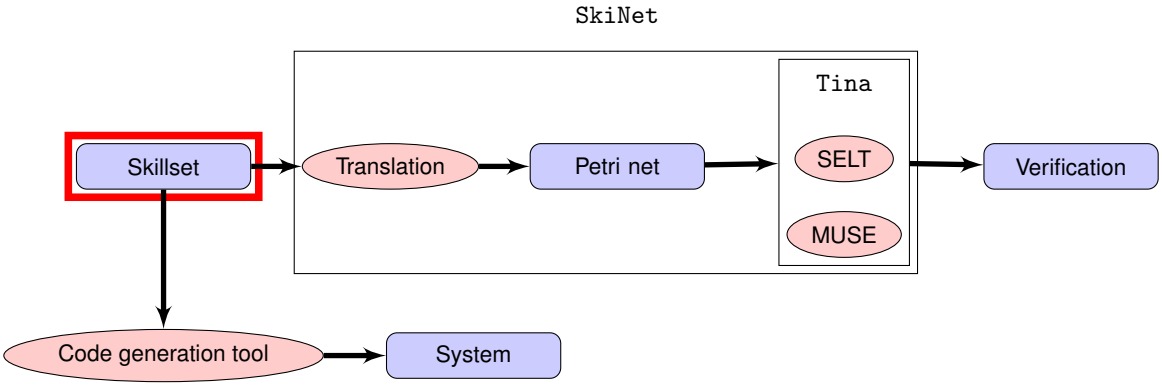
- How can we check whether our **high-level model** of the system is not flawed?
- Are there **undesired or blocking states**?
- Can it be polyvalent and perform a wide range of **missions**?
- Can we make the verification tool **easy to use and specify**?

SkiNet - Tool overview



[3] B. Berthomieu, P.-O. Ribet and F. Vernadat (2004): The tool TINA – Construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research 42, pp. 2741–2756, doi:10.1080/00207540412331312688.
[4] Bernard Berthomieu, François Vernadat and Silvano dal Zilio (2004): The TINA toolbox Home Page - Time Petri Net Analyzer - by LAAS/CNRS. Available at https://projects.laas.fr/tina/home.php

SkiNet - Skillset



Example system - Spot® skillset architecture

Skillset resources

- power_status: PowerOff PowerOn
- lease_status: AutoMode ManualMode
- control_mode: Idle Busy

Skills

- init_power
- safe_poweroff
- go_to

Resources

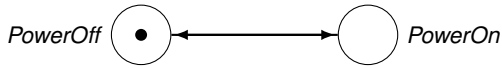
```
skillset spot_fmas {

    resource {
        power_status {
            initial PowerOff
            PowerOff -> PowerOn
            PowerOn -> PowerOff
        }
        lease_status {
            initial AutoMode
            AutoMode -> ManualMode
            ManualMode -> AutoMode
        }
        control_mode {
            initial Idle
            Idle -> Busy
            Busy -> Idle
        }
    }
```

Resources $\mathcal{R}$ = finite-state machines (FMSs)

- **Internal elements**: battery level, stance, power status, control mode, mutex, sensor data...
- **External elements**: operator control, environment objects, mission elements...

Ex. `power_status`:

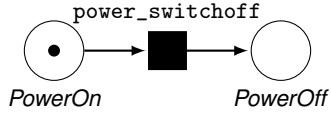


Events - Resource guards and effects

```
event {
    toauto_frommanual {
        guard lease_status == ManualMode
        lease_status -> AutoMode
    }
    tomanual_fromauto {
        guard lease_status == AutoMode
        lease_status -> ManualMode
    }
    power_switchoff {
        guard power_status == PowerOn
        power_status -> PowerOff
    }
    power_switchon {
        guard power_status == PowerOff
        power_status -> PowerOn
    }
}
```

Events $\mathcal{V}$ = **uncontrollable elements** that can affect the state of resources.

- **Resource guard** = **logical formula**
  $\phi : \{S^r, r \in \mathcal{R}\} \rightarrow \{True, False\}$
- **Effect** $\epsilon = (r, S_i^r)$ of a resource and its **next state**.

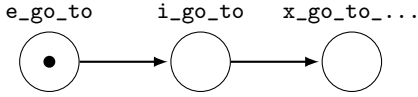


Skills

Skills $\mathcal{S}$ = **elementary actions** FSMs:

- An **id**.
- **States**: idle e_s, running i_s and ended $x_{s,k}$
- ...

```
skill go_to {
  ...
```

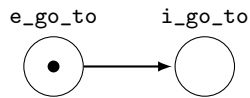


Skills

Skills $\mathcal{S}$ = **elementary actions** FSMs:

- ...
- **Preconditions** (start guards). Failure effects can happen when skill starts but guards aren't satisfied.
- **Start** effects if all preconditions are satisfied.
- ...

```
skill go_to {
    precondition {
        is_powered { guard (power_status==PowerOn) }
        is_auto { guard (lease_status==AutoMode) }
        is_idle { guard (control_mode==Idle) }
    }
    start control_mode -> Busy
    ...
```

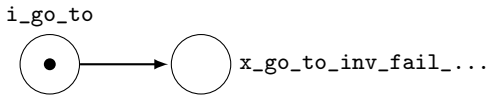


Skills

Skills $\mathcal{S}$ = elementary actions:

- ...
- **Invariants** (execution guards). Skill ends immediately upon failure, with failure effects.
- ...

```
skill go_to {
    ...
    invariant {
        is_powered {
            guard (power_status==PowerOn)
            failure control_mode -> Idle
        }
        is_auto {
            guard (lease_status==AutoMode)
            failure control_mode -> Idle
        }
    }
    ...
```

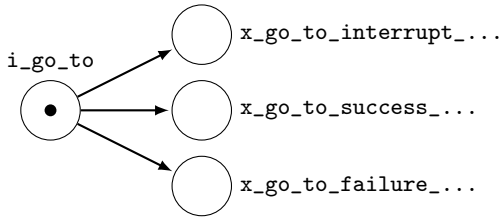


Skills

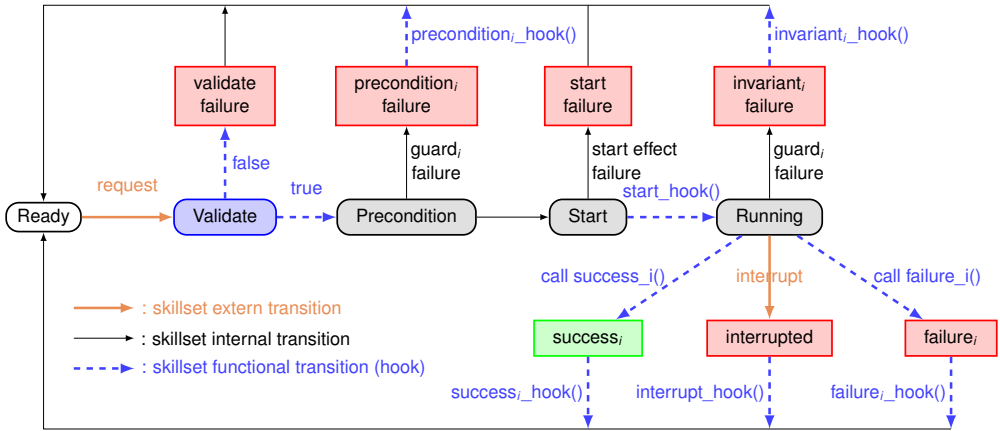
Skills $\mathcal{S}$ = elementary actions:

- ...
- **Interrupt** (external operation).
- **Success** and **Failure** (internal operations).

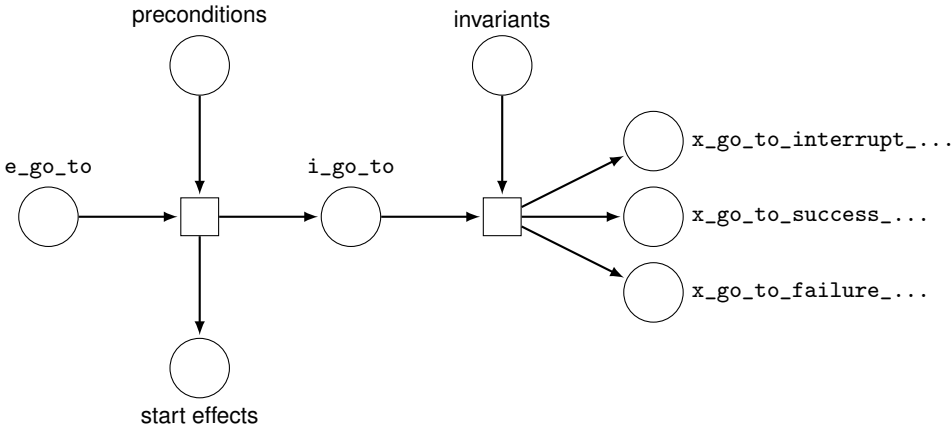
```
skill go_to {
    ...
    interrupt control_mode -> Idle
    success is_arrived control_mode -> Idle
    failure failed_reach control_mode -> Idle
}
```



Skills

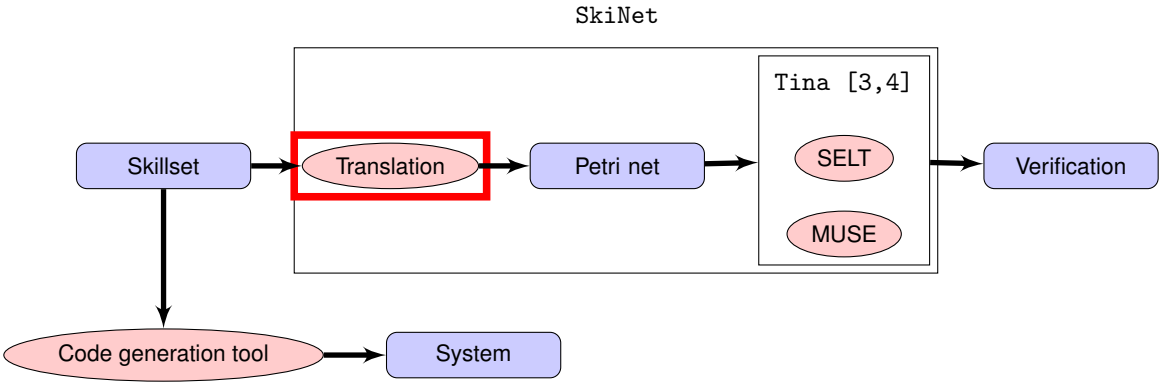

precondition$_i$_hook()
invariant$_i$_hook()
validate failure
precondition$_i$ failure
start failure
invariant$_i$ failure
false
guard$_i$ failure
start effect failure
guard$_i$ failure
request
true
start_hook()
Ready
Validate
Precondition
Start
Running
call success_i()
interrupt
call failure_i()
: skillset extern transition
: skillset internal transition
: skillset functional transition (hook)
success$_i$
interrupted
failure$_i$
success$_i$_hook()
interrupt_hook()
failure$_i$_hook()

Goal for the generated Petri net



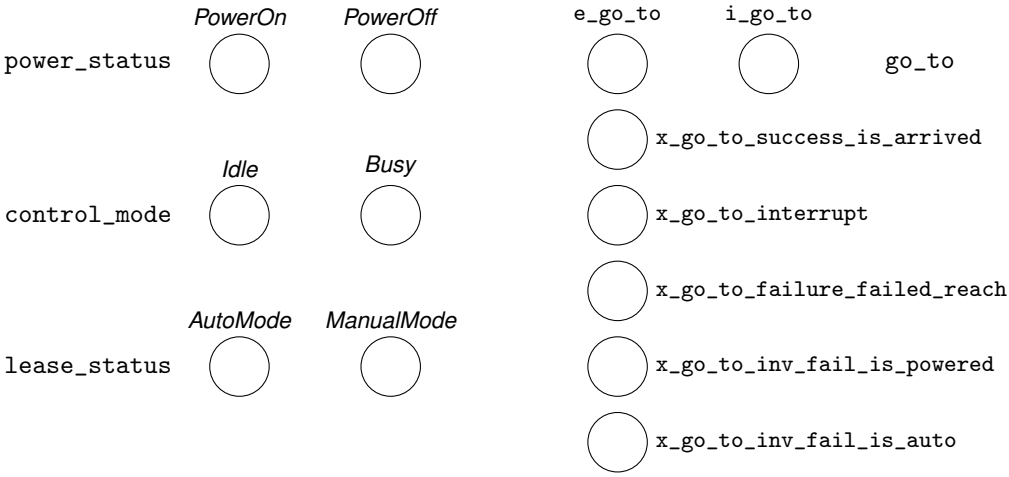
SkiNet - Translation

Petri nets = ideal for modelling **concurrent state-machines** with **shared resources**.



Translation - Places generation

Places $P = P_r$ resource state places $\cup$ P_s skills execution state



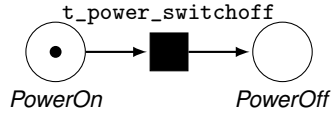
Translation - Transitions generation

We introduce **skillset transitions** $\tau = (\phi, \mathcal{E}, \sigma)$ to generate the transitions $T = \bigcup T_\tau$.
Skillset transition τ:

- guard ϕ
- effects $\mathcal{E}$
- state change σ

$$\begin{aligned} \tau_{\texttt{power\_switchoff}} = \{&\phi = (\texttt{power\_status} == \textit{PowerOn}), \\ &\mathcal{E} = ((\texttt{power\_status}, \textit{PowerOff})), \\ &\sigma = \varnothing\} \end{aligned} \tag{1}$$

```
event power_switchoff {
    guard power_status == PowerOn
    power_status -> PowerOff
}
```

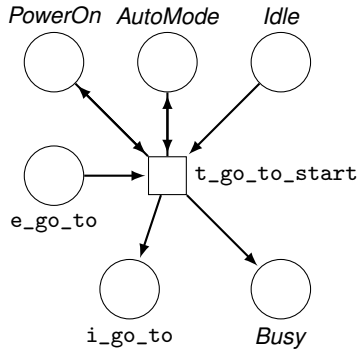


Translation - Transitions generation

```
skill go_to {
    precondition {
        is_powered { guard (power_status==PowerOn) }
        is_auto { guard (lease_status==AutoMode) }
        is_idle { guard (control_mode==Idle) }
    }
    start control_mode -> Busy
    ...
```

If a resource r is **evaluated** by ϕ but **not affected** by $\mathcal{E}$?
$\rightarrow$ **Reading arc**

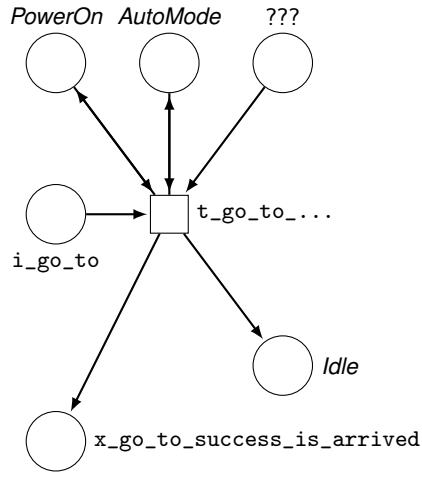
$$\begin{aligned}\tau_{\texttt{go\_to\_start}} = \{&\phi = (\texttt{power\_status} == \textit{PowerOn} \\ &\wedge \texttt{lease\_status} == \textit{AutoMode} \\ &\wedge \texttt{control\_mode} == \textit{Idle}), \\ &\mathcal{E} = ((\texttt{control\_mode}, \textit{Busy})), \\ &\sigma = e_{go\_to} \rightarrow i_{go\_to}\}\end{aligned} \tag{2}$$



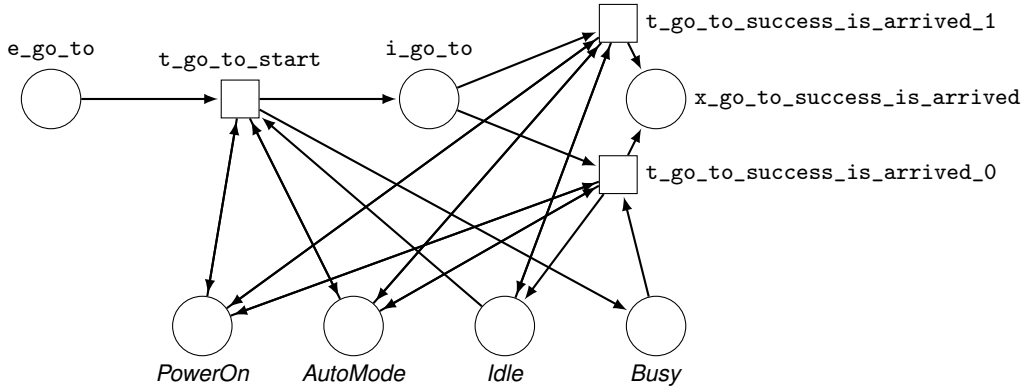
Translation - Transitions generation

```
skill go_to {
    ...
    invariant {
        is_powered {
            guard (power_status==PowerOn)
            failure control_mode -> Idle
        }
        is_auto {
            guard (lease_status==AutoMode)
            failure control_mode -> Idle
        }
    }
    interrupt control_mode -> Idle
    success is_arrived control_mode -> Idle
    failure failed_reach control_mode -> Idle
    }
```

If a resource r is **affected** by $\mathcal{E}$ but **not guarded** by ϕ?
$\rightarrow$ **Need to anticipate the possible origin states!**

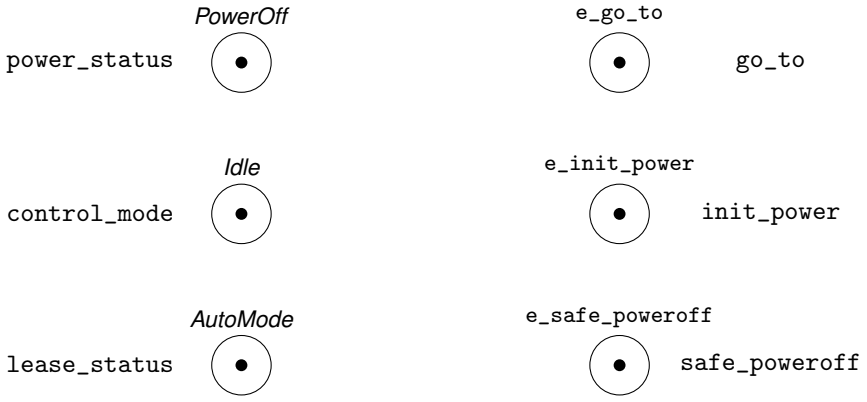


Translation - go_to start + success



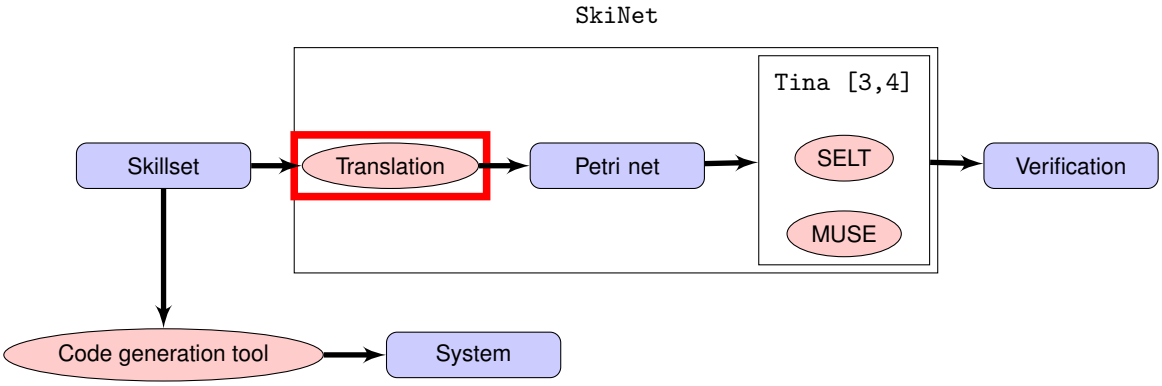
Initial marking

Resources in their **initial state**, skills are **idle**:



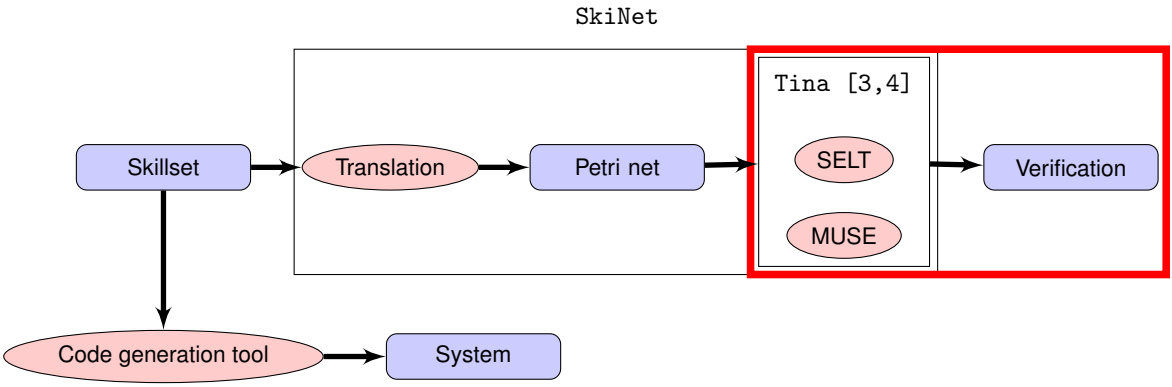
SkiNet - Translation

Generation is **formally verified** in the paper and compliant with skillset semantics.



SkiNet - Verification

After generating the **reachability graph** of the generated net, we can perform **model-checking**.



SELT - LTL Model-checker

Deadlocks $\rightarrow$ can the skillset block completely?

$$A\neg dead \tag{3}$$

Dead transitions $\rightarrow$ are there unnecessary elements?

$$\forall t \in T : A\ \neg t \tag{4}$$

SELT - LTL Model-checker

Check the FSM property of resources and skills (only state active at a time):
Resource invariants:

$$\forall r \in \mathcal{R} : A \left(\sum_{S_i^r \in S^r} p_i^r = 1 \right) \tag{5}$$

Skill invariants:

$$\forall s \in \mathcal{S} : A \left(p_e^s + p_i^s + \sum_k p_{x,k}^s = 1 \right) \tag{6}$$

In general, check if no states are stacking:
1-safeness

$$\forall p \in P : A \neg (p \geq 2) \tag{7}$$

MUSE - mu-calculus CTL model-checker

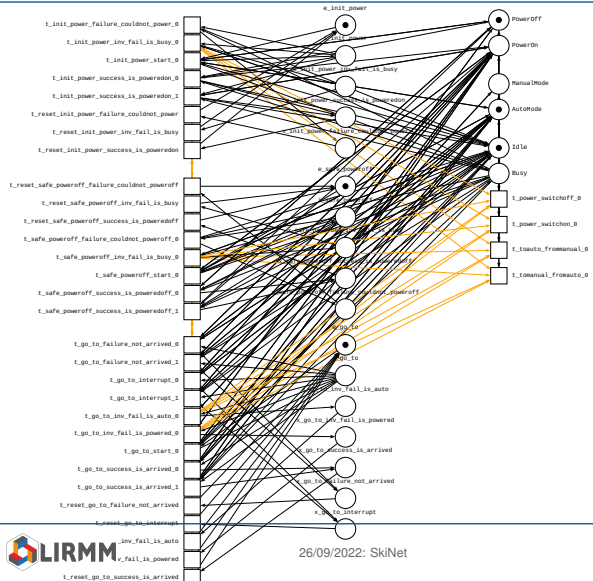
Skill liveness $\rightarrow$ Can a skill suddenly become inoperable forever?

$$\neg AFEF\ p_i^s, s \in \mathcal{S} \tag{8}$$

For all skills:

$$\neg AFEF\ (\sum_{s \in S} p_i^s) \tag{9}$$

Generated net - Spot example



Verification - Spot example

```
Analysis of the skillset net.
Loading .ktz file, this can take a few seconds...
Selt version 3.7.0 -- 01/19/22 -- LAAS/CNRS
ktz loaded, 375 states, 1648 transitions
Muse version 3.7.0 -- 01/11/22 -- LAAS/CNRS
ktz loaded, 375 states, 1648 transitions
Loading successful!
Basic check: [all/dead/live/safe/skill/deadset/quit]
No deadlock: [] -dead
TRUE
Transition is never fired: [] - t
7 dead transitions found:
t_init_power_inv_fail_is_busy_0, t_reset_init_power_inv_fail_is_busy, t_reset_safe_poweroff_inv_fail_is_busy, t_safe_poweroff_inv_fail_is_busy_0, t_go_to_failure_not_arrived_1, t_go_to_interrupt_1, t_go_to_success_is_arrived_1
No more than one token per resource and skills: [] (p1 + p2 + ...) = 1
power_status
TRUE
lease_status
TRUE
control_mode
TRUE
init_power
TRUE
safe_poweroff
TRUE
go_to
TRUE
No more than one token per place: [] -( p1 >= 2 \/ p2 >= 2 \/ ... )
TRUE
```

Verification - Spot example

```
Skill will always eventually die: -AF EF i_skill
--init_power--
FALSE, PATH TO DEAD STATE IS:
0 -> t_power_switchon_0
2 -> t_go_to_start_0
8 -> t_power_switchoff_0
Final state: 21

--safe_poweroff--
FALSE, PATH TO DEAD STATE IS:
Identical as previous (21).

--go_to--
FALSE, PATH TO DEAD STATE IS:
0 -> t_power_switchon_0
2 -> t_go_to_start_0
8 -> t_power_switchoff_0
21 -> t_go_to_inv_fail_is_powered_0
Final state: 49

Skillset will always eventually die: -AF EF (i_sk1 /\ i_sk2 /\ ...)
FALSE, PATH TO DEAD STATE IS:
0 -> t_power_switchon_0
2 -> t_go_to_start_0
8 -> t_power_switchoff_0
21 -> t_go_to_inv_fail_is_powered_0

Basic check: [all/dead/live/tokens/skill/deadset/quit]
```

Conclusion & Future work

- We developped a tool that can **verify the global behavior** of a system based on a **skillset model**.
- We can now **test and improve the model** before implementation until we're satisfied.
- However, **scalability issues** can arise if the model is too large...

If scalability issues arise, we are developping a **runtime verification** extension of the tool that generate **rolling partial state-spaces** of the Petri net model and checks the **satisfaction of specifications**.

We also work on a **planification** extension to allow users to verify and send **high-level goals** to the system, using **sensor data**.

Git repository for the tool: https://gitlab.com/onera-robot-skills/skinet-release

Petri net

Why Petri nets?

Petri nets are very intuitive to use when it comes to **modelling concurrent finite state-machines and shared resources**. They are a simple mathematical tool and a lot of tools to manipulate them are available online.

A **Petri net** $\langle N, m_0 \rangle$ is a tuple $N = (P, T, F)$ and an initial marking m_0, where:

- P and T are two non-empty, disjoint and finite sets of **places and transitions**, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$ a set of **directed arcs**.
- $m_0 \in M$ the **initial distribution of tokens**, called the initial marking of the net, and $M = \{m_0, m_1, ..., m_n\}$ the set of **all possible markings** of N.

This definition is **extended** with $N = (P, T, F, \succ)$, where $\succ$ is the **priority relation**, represented by directed arcs between transitions, with the source transition having a higher priority.

For a transition $t \in T$, we note ${}^\bullet t$ and $t^\bullet$ its **input and output places** respectively.

The **marking of a place** p is noted $m[p]$. The **firing of an enabled transition** $t \in T$, with $\forall p \in^\bullet t,\ m[p] \geq 1$, leads to a new marking, or reachable state, m'. All the **input places** of t **loose a token**, i.e. $\forall p \in^\bullet t,\ m'[p] = m[p] - 1$, and all the **output places gain one token**, i.e. $\forall p \in t^\bullet,\ m'[p] = m[p] + 1$.