TTEthernet alprotokoll vizsgálata [1]

Szoftver verifikáció és validáció kiselőadás

Ferencz Bálint | ferencz@mit.bme.hu

Itíner

- Óraszinkronizációról általában
- TTEthernet gyorstalpaló
- TTE *compression* algoritmus bizonyítása
  - Formális modellje
  - Modellellenőrzés lépései

Óraszinkronizáció

Miért?

Aszinkron rendszerek halmazán szeretnénk szinkron tulajdonságokat definiálni

Konkrét példák

- Szeretnénk minden kijelzőn ugyanazt a kijelzett időt látni.
- TLS tanúsítványok érvényességében biztosak szeretnénk lenni.
- Egy erőátviteli telephelyen kíváncsiak lehetünk a hálózati feszültségekre, áramokra.
- Nagy fizikai kísérleteknél is alapvető feladat!

Óraszinkronizáció a gyakorlatban

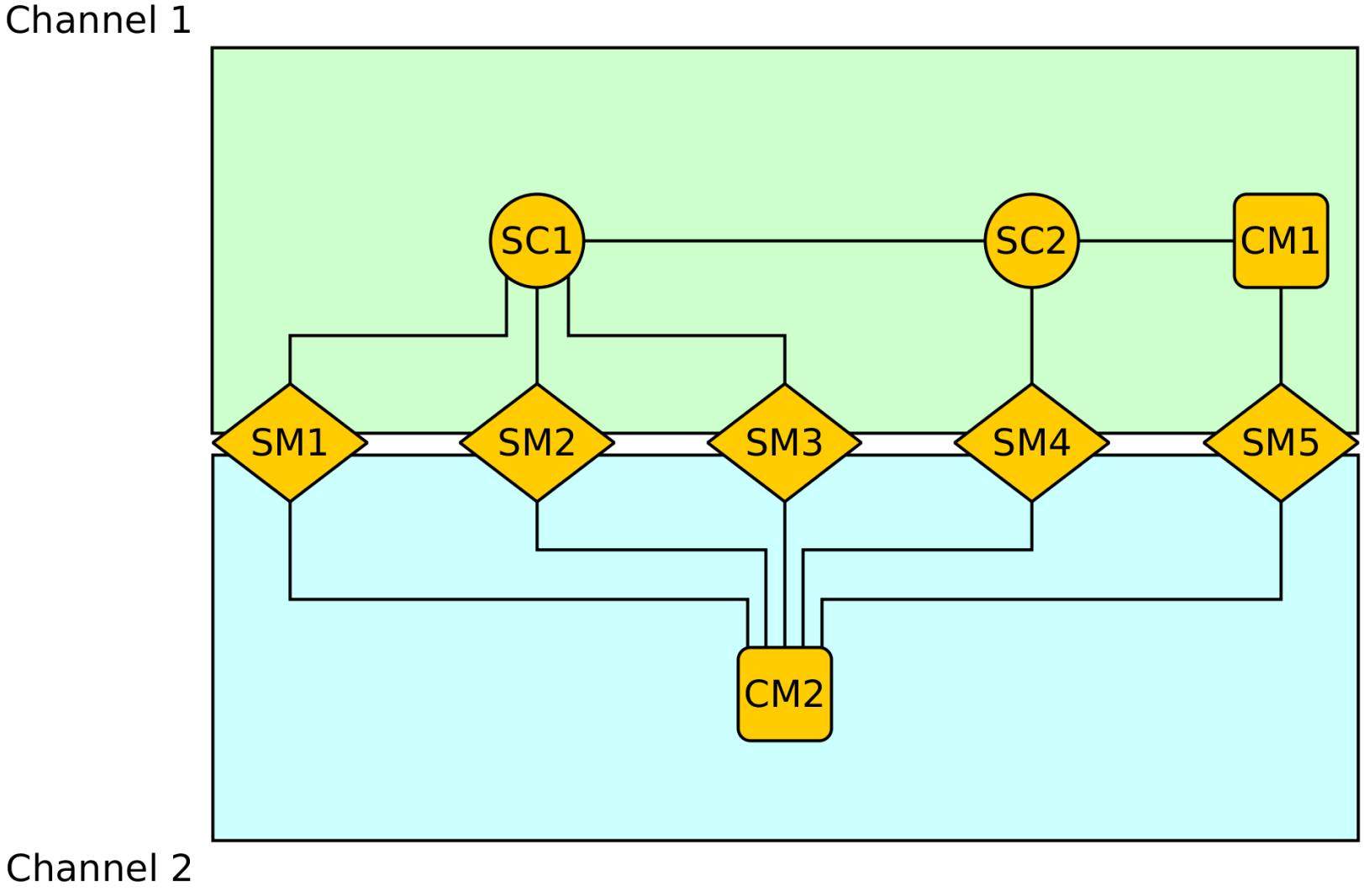
- Kézi módszerrel
- Dedikált vezetékezéssel
- A már kiépített kommunikációs médiumon
  - NTP
  - PTP (IEEE 1588)
  - White Rabbit
  - **Time-Triggered Ethernet (SAE AS6802)**

Mi az a TTEthernet?

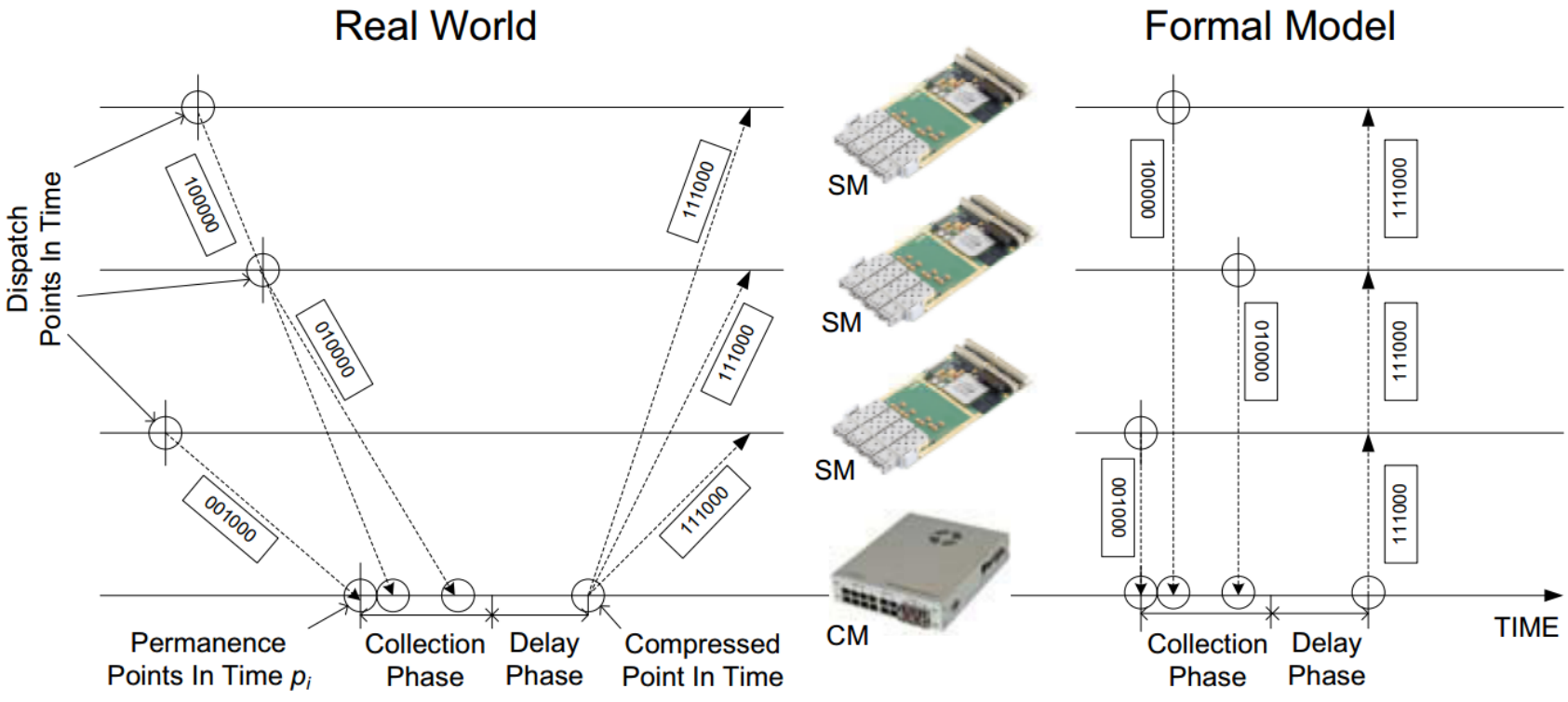
- Hozzuk be az Ethernetet a biztonságkritikus rendszerekbe - korlátokkal!
- Kritikus üzenetek csak eltárolt üzenetváltási időslotokban forgalmazhatóak
  - **Szinkronban kell járniuk az óráknak!**
- Miért szükséges a formális bizonyítás?


787
DREAMLINER
BOEING

Mi az a TTEthernet?



Mi az a TTEthernet?



Protokollbizonyítás általánosságban

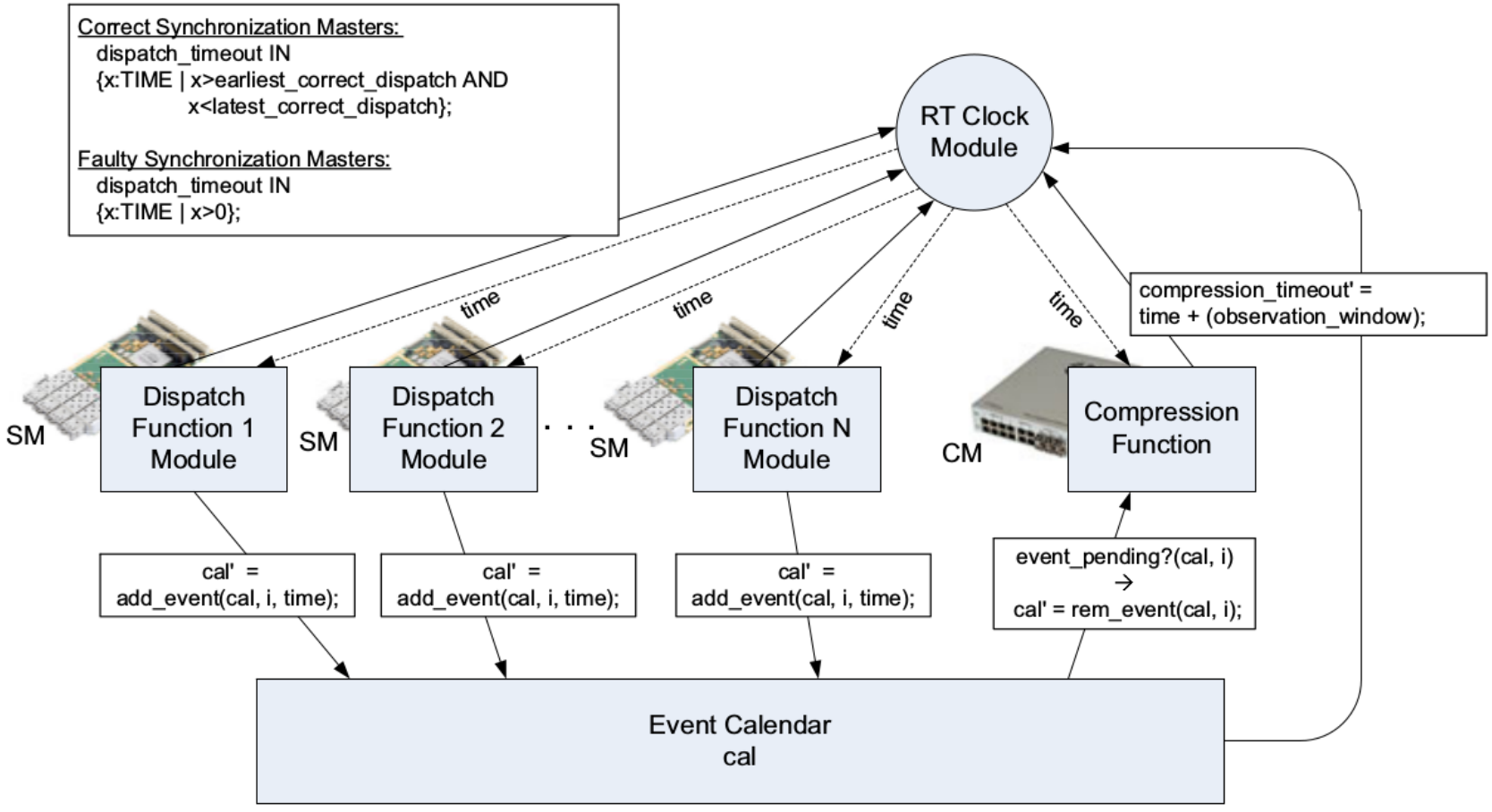
Alapprobléma

- Adottak az aszinkron modellek.
- A modellek halmazának együttes viselkedése érdekes.
- Hogyan bizonyítjuk?

TTEthernet formális modell

Naptár automata [2]

TTEthernet formális modell



TTEthernet formális modell

Általános konstansok

```
k: NATURAL = 3; # Max. hibás node-ok száma
N: NATURAL = 3*k + 1; # Összes node száma
OBSERVATION_WINDOW_ID: TYPE = [1..k*3];
```

Megfigyelési ablak tulajdonságai

```
observation_window: REAL = 5; # Fix idő paraméter
earliest_correct_dispatch: REAL=
        (k+1)*observation_window;
latest_correct_dispatch: REAL =
        earliest_correct_dispatch + observation_window;
end_of_time: REAL = latest_correct_dispatch +
        ((k+1)+2)*observation_window;
```

TTEthernet formális modell

Időkezelés

```
clock_readings: TYPE =
[# valid: ARRAY DISPATCH_ID OF BOOLEAN,
   value: ARRAY DISPATCH_ID OF TIME #];

add_clock_reading(cr: clock_readings,
                  i: DISPATCH_ID,
                  v: TIME): clock_readings =
((cr WITH .valid[i] := TRUE) WITH .value[i] := v);

%use ft_median for k=1, N=4
 ft_median4(cr: clock_readings): TIME =
   IF    cr.valid[4] THEN (cr.value[2] + cr.value[3])/2
   ELSIF cr.valid[3] THEN cr.value[2]
   ELSIF cr.valid[2] THEN (cr.value[1]+cr.value[2]) / 2
   ELSE                    cr.value[1]
   ENDIF;
```

TTEthernet formális modell

> Ez itt további 5 oldalnyi cikk és 500 sor SAL kód hűlt helye

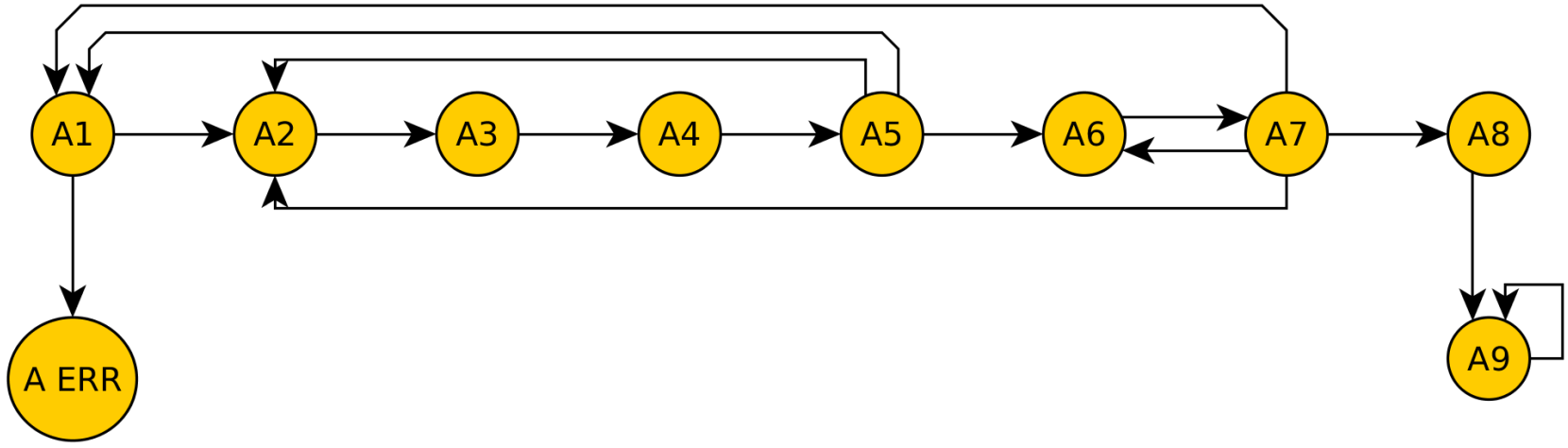
Mit szeretnénk ellenőrizni a modellen?

Négy alaptulajdonságot

- *agreement*: ha a megfigyelési ablakon belül vettünk egy érvényes PCF-et, akkor ue. megfigyelési ablakon belül az összes helyesen működő SM-től is venni fogunk
- *window*: a számított idő a $[k \cdot obs\_win, (k+2) \cdot obs\_win]$ intervallumban található
- *correction*: minden jól működő SM által vett *"korrigált idő"* eltérése a saját órájuktól kevesebb, mint obs_win
- *termination*: az algoritmus véges idő alatt eredményt nyújt

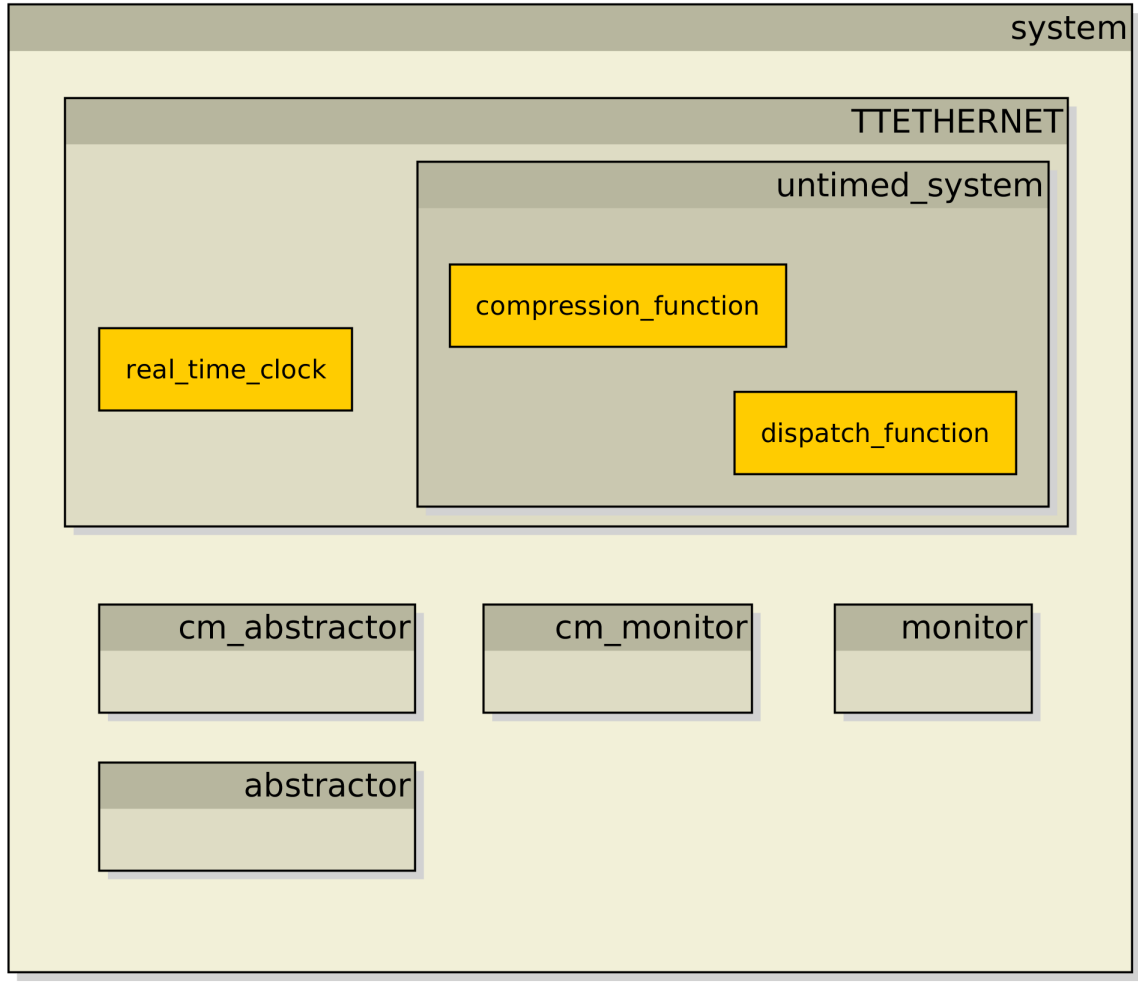
Hogyan verifikáljuk a modellt?

Absztrakció segítségével



Sajnos az automatikus k-indukció itt nem működik

Komponált modell részlet



"Agreement" invariáns

```
reading_index =
IF count_msg(1, cal, list_dispatch_states,
             list_dispatch_pits, pit_0,
             window_counter, old_values) < N
THEN count_msg(1, cal, list_dispatch_states,
             list_dispatch_pits, pit_0,
             window_counter, old_values) + 1
ELSE N ENDIF
```

Akkor teljesül, ha a ugyanannyi SM bocsátott már ki PCF-et, mint amennyit a `reading_index` CM számlálója mutat.
A `count_msg` függvény megszámolja azokat az SM-eket, amik adtak ki PCF-et **ÉS** a CM felhasználta már azokat.

Időzítési tulajdonságok invariánsai

"Window" és "correction"

Nehezebb, mert fel kell vázolnunk az SM-ek üzenetküldési szekvenciáját.

```
(FORALL (i:DISPATCH_ID):
IF i = 1 THEN clock_stack.value[i]=0
ELSIF i <= count_memb(1, membership_new)
THEN clock_stack.value[i] =
        list_dispatch_pits[observed_order[i]] -
        list_dispatch_pits[observed_order[1]]
ELSE clock_stack.value[i]=0 ENDIF)
```

Majd rendeznünk kell is a stacket, amire a cikk felvázol egy trükköt (tömbrendezés helyett rendezett "pointertömb" készítése).

Verifikációs lemmák

```
agreement: LEMMA system |-
 G(compression_state=cm_compressed =>
(FORALL (i:DISPATCH_ID): i<=k OR membership_new[i]));
```

Ez azt jelenti, hogyha elértük a CM végállapotát, akkor akkor minden jól működő SM jelezve van a `membership_new` vektorban.

Verifikációs lemmák

```
window: LEMMA system |-
  G(compression_state=cm_compressed
  AND compressed_true =>
(FORALL (i:DISPATCH_ID): i<=k OR
( list_dispatch_pits[i]+k*observation_window <=
  time_out[COMPRESSION_FUNCTION_ID]
 AND
  time_out[COMPRESSION_FUNCTION_ID] <=
  list_dispatch_pits[i]+(k+2)*observation_window)
)
);
```

Ez azt állítja, hogy a CM által kibocsátott korrigált idő értéke korlátos.

Verifikációs lemmák

```
correction: LEMMA system |-
G(compression_state=cm_compressed
 AND compressed_true =>
(FORALL (i:DISPATCH_ID): i<=k OR
(time_out[COMPRESSION_FUNCTION_ID] -
 list_dispatch_pits[i] +
 (k+1)*observation_window <= observation_window)
OR
(list_dispatch_pits[i] +
 (k+1)*observation_window -
 time_out[COMPRESSION_FUNCTION_ID]
 <= observation_window)));
```

Akkor teljesül, ha ha a CM által kibocsátott *permanence time*-ra igaz az, hogy maximum *observation window*-nyit tér el a vevő órák állásától.

Verifikációs lemmák

```
termination: LEMMA system |-
                    G(compression_state/=cm_error);
```

Előséget jelöl, azaz soha nem lépünk hibaállapotba, a compression algoritmusnak van kimenete, és a bizonyító nem áll meg a triviális bizonyításoknál.

```
abstract_inv: LEMMA system |- G(state/=bad);
```

Szintén élőségi kritérium, hogy az absztrakt modellünk soha nem lép a hibaállapotába.

Modellellenőrzés végrehajtása


```
sal-inf-bmc
```

The SAL infinite bounded model checker; this is a model checker for infinite state systems based on SMT solving. In addition to refutation (i.e., bug detection and counterexample generation), the SAL infinite bounded model checker can perform verification by k-induction. SAL can use several SMT solvers, but defaults to ***Yices****.*

Eredmények

Tulajdonság	k	N	T_{verify}		k	N	T_{verify}
agreement	1	4	1.65 sec		2	7	2.58 sec
window	1	4	1.81 sec		2	7	7.49 sec
correction	1	4	1.80 sec		2	7	4.07 sec
termination	1	4	1.72 sec		2	7	2.67 sec
abstract_inv	1	4	6.51 sec		2	7	2 227.38 sec

- $k >= 3$ esetén órákig futott eredmény nélkül
- Ez nem gond, mivel a szabvány $k = 2$ hibás node-ot engedélyez

Mi a tanulság?

A modellek mellett absztrakt modelleket is gyártani kellett, ami nem triviális feladat

Több kiegészítő lemma szükséges

- idő: 7 db
- timeout: 4 db
- dispatch: 4 db
- hibás node: 2 db
- absztrakt modell: 5 db

Németek által írt angol nyelvű cikkek fárasztóak.

Köszönöm a figyelmet!

Források

- [1] Steiner, Wilfried, and Bruno Dutertre. "SMT-based formal verification of a TTEthernet synchronization function." International Workshop on Formal Methods for Industrial Critical Systems. Springer Berlin Heidelberg, 2010.
- [2] Dutertre, Bruno, and Maria Sorea. "Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata." Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. Springer Berlin Heidelberg, 2004. 199-214.