

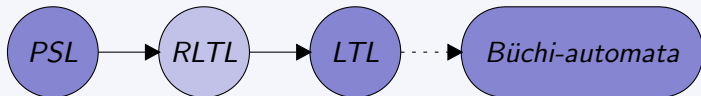
Deep Embeddings of Temporal Logics in HOL

Thomas Tuerk

ARG Lunch, 21th June 2006

Original situation

- in my diploma thesis I formally validated a translation of a significant subset of *PSL* to generalised Büchi automata
- the emptiness problem of generalised Büchi automata remained

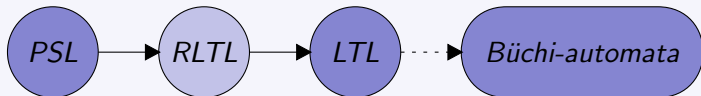


Goal

Create an automatic tool inside *HOL* to model check *PSL* formulae!

Original situation

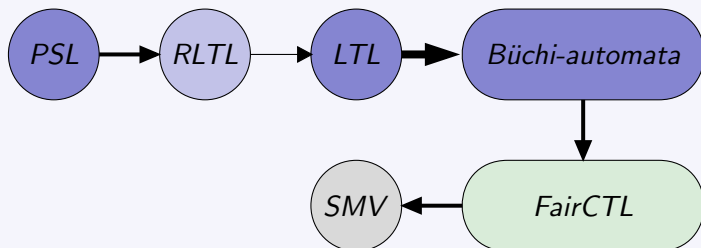
- in my diploma thesis I formally validated a translation of a significant subset of *PSL* to generalised Büchi automata
- the emptiness problem of generalised Büchi automata remained



Goal

Create an automatic tool inside *HOL* to model check *PSL* formulae!

Work done



- automatise the translation of *LTL* to Büchi-automata
- optimise the translation of *PSL* to *RLTL*
- create an interface to *SMV*

Work done II

- deep embeddings of important formalisms for model checking:
 - propositional logic
 - *LTL*
 - *RLTL*
 - *CTL*, *FairCTL*, *CTL**
 - kripke structures, symbolically and explicitly represented
 - automaton formulae, i. e. symbolically represented
 - nondeterministic and universal ω -automata
 - explicitly represented nondeterministic and universal ω -automata
 - alternating ω -automata
- application: tool for IBM

Overview

- 1 Introduction
- 2 Examples
- 3 Extension of the Simplifier
- 4 Application: Tool for IBM
- 5 Conclusions

Translation of *LTL* to Automata

LTL formula

$$(XX a) \wedge G(\neg a)$$

At the next but one point of time *a* holds and *a* never holds!

```
> val ltl =
  'LTL_AND
    (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
     LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a))))';
```

Translation to Generalized Büchi-Automata

```

- ltl2omega true true ltl;

> val it =
  []
  |- !sv.
    IS_ELEMENT_ITERATOR sv 3 {a} ==>
    !i.
      LTL_SEM i
        (LTL_AND
          (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
            LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a)))))) =
      A_SEM i
        (A_NDET
          (symbolic_semi_automaton (\s. ?n. n < 3 /\ (s = sv n)) (P_BIGAND []))
            (XP_BIGAND
              [XP_EQUIV (XP_PROP (sv 1),XP_NEXT (P_PROP (sv 0)));
                XP_EQUIV (XP_PROP (sv 0),XP_NEXT (P_PROP a));
                XP_EQUIV
                  (XP_PROP (sv 2),
                    XP_OR
                      (XP_CURRENT (P_PROP a),
                        XP_AND (XP_CURRENT (P_NOT P_FALSE),XP_NEXT_PROP (sv 2)))))]),
              A_AND
                (A_BIGAND [],
                  ACCEPT_COND_PROP (P_AND (P_PROP (sv 1),P_NOT (P_PROP (sv 2)))))))

```


Translation to Generalized Universal Persistency Automata

```

- ltl2omega true false ltl;

> val it =
  []
  |- !sv.
    IS_ELEMENT_ITERATOR sv 3 {a} ==>
    !i.
      LTL_SEM i
        (LTL_AND
          (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
            LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a)))))) =
      A_SEM i
        (A_UNIV
          (symbolic_semi_automaton (\s. ?n. n < 3 /\ (s = sv n)) (P_BIGAND []))
            (XP_BIGAND
              [XP_EQUIV (XP_PROP (sv 1),XP_NEXT (P_PROP (sv 0)));
                XP_EQUIV (XP_PROP (sv 0),XP_NEXT (P_PROP a));
                XP_EQUIV
                  (XP_PROP (sv 2),
                    XP_OR
                      (XP_CURRENT (P_PROP a),
                        XP_AND (XP_CURRENT (P_NOT P_FALSE),XP_NEXT_PROP (sv 2)))))]),
              A_IMPL
                (A_BIGAND [ACCEPT_COND_GF (P_IMPL (P_PROP (sv 2),P_PROP a))],
                  ACCEPT_COND_PROP (P_AND (P_PROP (sv 1),P_NOT (P_PROP (sv 2)))))))))

```

Exploiting Termsharing

Examples of a counter with $n + 1$ bits (kindly provided by Kristin Rozier).

	fast		termsharing	
1	3.9 s	24 state vars	11.7 s	13 state vars
2	5.1 s	34 state vars	14.6 s	17 state vars
3	6.5 s	44 state vars	18.4 s	21 state vars
4	7.9 s	54 state vars	23.1 s	25 state vars
5	9.5 s	64 state vars	28.8 s	29 state vars
...				
10	18.8 s	114 state vars	76.7 s	49 state vars
15	30.4 s	164 state vars	167.8 s	69 state vars
20	43.9 s	214 state vars	315.3 s	89 state vars
25	59.1 s	264 state vars	533.1 s	109 state vars

Translation to Fair Emptiness Check I

```
- ltl_contradiction2ks_fair_emptiness true ltl;

> val it =
  []
  |- !sv.
    IS_ELEMENT_ITERATOR sv 3 {a} ==>
    (LTL_IS_CONTRADICTION
     (LTL_AND
      (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
       LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a)))))) =
    IS_EMPTY_FAIR_SYMBOLIC_KRIPKE_STRUCTURE
    (symbolic_kripke_structure (P_AND (P_PROP (sv 1),P_NOT (P_PROP (sv 2))))
     (XP_AND
      (XP_EQUIV (XP_PROP (sv 2),XP_OR (XP_PROP a,XP_NEXT_PROP (sv 2))),
       XP_AND
        (XP_EQUIV (XP_PROP (sv 1),XP_NEXT_PROP (sv 0)),
         XP_EQUIV (XP_PROP (sv 0),XP_NEXT_PROP a)))))) [] : thm
```

Translation to Fair Emptiness Check II

```
- ltl_contradiction2ks_fair_emptiness__num 1 true ltl;

> val it =
  ( []
    |- ALL_DISTINCT [a] ==>
      (LTL_IS_CONTRADICTION
        (LTL_AND
          (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
            LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a)))))) =
        IS_EMPTY_FAIR_SYMBOLIC_KRIPKE_STRUCTURE
          (symbolic_kripke_structure (P_AND (P_PROP 1,P_NOT (P_PROP 2)))
            (XP_AND
              (XP_EQUIV (XP_PROP 1,XP_NEXT_PROP 0),
                XP_AND
                  (XP_EQUIV (XP_PROP 0,XP_NEXT_PROP 3),
                    XP_EQUIV (XP_PROP 2,XP_OR (XP_PROP 3,XP_NEXT_PROP 2)))))) []),

    [(3, 'a')] : thm * (int * term) list
```

Translation to Fair Emptiness Check III

```
- ltl_contradiction2ks_fair_emptiness___num 2 true ltl;

> val it =
  ( []
  |- IS_EMPTY_FAIR_SYMBOLIC_KRIPKE_STRUCTURE
    (symbolic_kripke_structure (P_AND (P_PROP 1,P_NOT (P_PROP 2)))
      (XP_AND
        (XP_EQUIV (XP_PROP 1,XP_NEXT_PROP 0),
          XP_AND
            (XP_EQUIV (XP_PROP 0,XP_NEXT_PROP 3),
              XP_EQUIV (XP_PROP 2,XP_OR (XP_PROP 3,XP_NEXT_PROP 2)))))) [] ==>
  LTL_IS_CONTRADICTION
    (LTL_AND
      (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
        LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a))))),

  [(3, 'a')] : thm * (int * term) list
```

Successful Model Checking

```
- model_check__ltl_contradiction ltl;

> val it =
  SOME []
    |- LTL_IS_CONTRADICTION
      (LTL_AND
        (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
          LTL_ALWAYS (LTL_NOT (LTL_PROP (P_PROP a)))))
  : thm option
```

Counter Example

```
- val ltl2 =
  ``LTL_AND (LTL_NEXT (LTL_NEXT (LTL_PROP (P_PROP a))),
    LTL_ALWAYS (LTL_PROP (P_PROP a)))``

- model_check__ltl_contradiction ltl2;
```

Formula is not true! Consider the countermodel:

```
=====
===== A loop starts here=====
===== State0=====
x1 = 1, x2 = 0, x3 = 1, x4 = 1
=====
```

resources used:

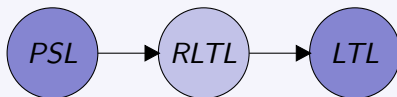
```
user time: 0 s, system time: 0 s
BDD nodes allocated: 357
Bytes allocated: 917504
BDD nodes representing transition relation: 33 + 1
reachable states: 8 (2^3) out of 16 (2^4)
```

```
=====
x1: sv 1, x2: sv 2, x3: sv 0, x4: a
=====
> val it = NONE : thm option
```

Other LTL Model Checking Problems

- $\forall i, t. i \not\models^0 ltl$
there is no path that satisfies ltl , i. e. ltl is a contradiction
- $M \models ltl$
the kripke structure M models ltl
- $\forall i, t. i \models^t ltl_1 \Leftrightarrow i \models^t ltl_2$
 ltl_1 and ltl_2 are equivalent
- $\forall i. i \models^0 ltl_1 \Leftrightarrow i \models^0 ltl_2$
 ltl_1 and ltl_2 are initially equivalent

Translation of *PSL* to *LTL*



Important Theorems

□ |- !f v b t.

IS_INFINITE_PROPER_PATH v /\ F_CLOCK_SERE_FREE f /\ ~(t = b) /\
 PATH_PROP_FREE t v /\ PATH_PROP_FREE b v ==>

(UF_SEM v f =

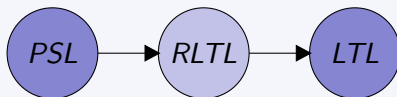
LTL_SEM (CONVERT_PATH_PSL_LTL t b v)

(RLTL_TO_LTL (P_PROP t) (P_PROP b) (PSL_TO_RLTL f)))

□ |- !M f. F_CLOCK_SERE_FREE f ==> (UF_KS_SEM M f = LTL_KS_SEM M (PSL_TO_LTL f))

□ |- !M f c. F_SERE_FREE f ==> (F_KS_SEM M c f = LTL_KS_SEM M (PSL_TO_LTL_CLOCK c f))

Translation of *PSL* to *LTL*



Important Theorems

- ```

[] |- !f v b t.
 IS_INFINITE_PROPER_PATH v /\ F_CLOCK_SERE_FREE f /\ ~(t = b) /\
 PATH_PROP_FREE t v /\ PATH_PROP_FREE b v ==>

 (UF_SEM v f =
 LTL_SEM (CONVERT_PATH_PSL_LTL t b v)
 (RTL_TO_LTL (P_PROP t) (P_PROP b) (PSL_TO_RLTL f)))

[] |- !M f. F_CLOCK_SERE_FREE f ==> (UF_KS_SEM M f = LTL_KS_SEM M (PSL_TO_LTL f))

[] |- !M f c. F_SERE_FREE f ==> (F_KS_SEM M c f = LTL_KS_SEM M (PSL_TO_LTL_CLOCK c f))

```

# Problems with the Translation of *PSL* to *LTL*

## 1st step: *PSL* to *RLTL*

```

[] |- PSL_TO_RLTL
 (F_ALWAYS
 (F_IMPLIES
 (F_STRONG_BOOL (B_PROP aa),
 F_STRONG_NEXT_EVENT
 (B_PROP bb,
 F_STRONG_BEFORE (F_STRONG_BOOL (B_PROP cc), F_STRONG_BOOL (B_PROP dd)))))) =
 RLTL_NOT
 (RLTL_SUNTIL
 (RLTL_PROP P_TRUE,
 RLTL_NOT (RLTL_NOT
 (RLTL_AND
 (RLTL_NOT (RLTL_NOT (RLTL_PROP (P_PROP aa))),
 RLTL_NOT (RLTL_SUNTIL
 (RLTL_PROP (P_NOT (P_PROP bb)),
 RLTL_AND
 (RLTL_PROP (P_PROP bb),
 RLTL_SUNTIL
 (RLTL_NOT (RLTL_PROP (P_PROP dd))),
 RLTL_AND
 (RLTL_PROP (P_PROP cc),
 RLTL_NOT (RLTL_PROP (P_PROP dd))))))))))))))

```

# Problems with the Translation of *PSL* to *LTL*

## 1st step: *PSL* to *RLTL*

```

[] |- PSL_TO_RLTL
 (F_ALWAYS
 (F_IMPLIES
 (F_STRONG_BOOL (B_PROP aa),
 F_STRONG_NEXT_EVENT
 (B_PROP bb,
 F_STRONG_BEFORE (F_STRONG_BOOL (B_PROP cc), F_STRONG_BOOL (B_PROP dd)))))) =
 RLTL_NOT
 (RLTL_SUNTIL
 (RLTL_PROP P_TRUE,
 RLTL_NOT (RLTL_NOT
 (RLTL_AND
 (RLTL_NOT (RLTL_NOT (RLTL_PROP (P_PROP aa))),
 RLTL_NOT (RLTL_SUNTIL
 (RLTL_PROP (P_NOT (P_PROP bb)),
 RLTL_AND
 (RLTL_PROP (P_PROP bb),
 RLTL_SUNTIL
 (RLTL_NOT (RLTL_PROP (P_PROP dd)),
 RLTL_AND
 (RLTL_PROP (P_PROP cc),
 RLTL_NOT (RLTL_PROP (P_PROP dd))))))))))))))

```

## Problems with the Translation of *PSL* to *LTL* II

### 2nd step: *RLTL* to *LTL*

```

[] |- RLTL_TO_LTL P_FALSE P_FALSE ... =
 LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_TRUE,P_NOT P_FALSE))),
 LTL_NOT (LTL_NOT (LTL_AND
 (LTL_NOT (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP aa,P_NOT P_FALSE))))),
 LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_NOT (P_PROP bb),P_NOT P_FALSE))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP bb,P_NOT P_FALSE))),
 LTL_SUNTIL
 (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP cc,P_NOT P_FALSE))),
 LTL_NOT
 (LTL_PROP
 (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))))))))))))))

```

- result of the translation is unnecessary large
- simple simplification steps possible

# Simplification example

## 3rd step: Simplification

```

[] |- LTL_EQUIVALENT
 (LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_TRUE,P_NOT P_FALSE))),
 LTL_NOT (LTL_NOT (LTL_AND
 (LTL_NOT (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP aa,P_NOT P_FALSE))))),
 LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_NOT (P_PROP bb),P_NOT P_FALSE))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP bb,P_NOT P_FALSE))),
 LTL_SUNTIL
 (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP cc,P_NOT P_FALSE))),
 LTL_NOT
 (LTL_PROP
 (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))))))))))))))
 (LTL_BEFORE (LTL_FALSE, LTL_AND
 (LTL_PROP (P_PROP aa),
 LTL_BEFORE
 (LTL_PROP (P_PROP bb),
 LTL_AND
 (LTL_PROP (P_PROP bb),
 LTL_SUNTIL
 (LTL_PROP (P_NOT (P_PROP dd)),
 LTL_PROP (P_AND (P_PROP cc,P_NOT (P_PROP dd))))))))))

```

# Simplification example

## 3rd step: Simplification

□ 1- **LTL\_EQUIVALENT**

```
(LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_TRUE,P_NOT P_FALSE))),
 LTL_NOT (LTL_NOT (LTL_AND
 (LTL_NOT (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP aa,P_NOT P_FALSE))))),
 LTL_NOT (LTL_SUNTIL
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_NOT (P_PROP bb),P_NOT P_FALSE))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP bb,P_NOT P_FALSE))),
 LTL_SUNTIL
 (LTL_NOT (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))),
 LTL_AND
 (LTL_PROP (P_OR (P_FALSE,P_AND (P_PROP cc,P_NOT P_FALSE))),
 LTL_NOT
 (LTL_PROP
 (P_OR (P_FALSE,P_AND (P_PROP dd,P_NOT P_FALSE))))))))))))))
(LTL_BEFORE (LTL_FALSE, LTL_AND
 (LTL_PROP (P_PROP aa),
 LTL_BEFORE
 (LTL_PROP (P_PROP bb),
 LTL_AND
 (LTL_PROP (P_PROP bb),
 LTL_SUNTIL
 (LTL_PROP (P_NOT (P_PROP dd)),
 LTL_PROP (P_AND (P_PROP cc,P_NOT (P_PROP dd))))))))))
```

## Extension of the Simplifier

- simplification according to semantic equivalence needed
  - HOL simplifier could only handle equality
  - but: underlying traverser could already handle arbitrary preorders (reflexive, transitive relations)
  - semantic equivalence is a congruence relation and therefore a preorder
- ⇒ create an additional interface of the traverser to use arbitrary preorders



## Extension of the Simplifier II

- `congLib` allows the usage of arbitrary preorders
- theorems have to be provided that
  - show that the relation is really a preorder
  - define congruence rules
  - define rewrite rules
- all these theorems can be combined into `congsetfrags` and `congsets`, similar to `simpsetfrags` and `simpsets` respectively
- simplifications and tactics exist that use these `congsets`
- no conditional rewriting and no ordered rewriting available

## Creating a Preorder

```

> val PROP_LOGIC_EQUIVALENT_REFL =
 [] |- !x. PROP_LOGIC_EQUIVALENT x x : thm

> val PROP_LOGIC_EQUIVALENT_TRANS =
 [] |- !x y z.
 PROP_LOGIC_EQUIVALENT x y /\ PROP_LOGIC_EQUIVALENT y z ==>
 PROP_LOGIC_EQUIVALENT x z : thm

- val prop_logic_equivalent_preorder =
 mk_preorder (PROP_LOGIC_EQUIVALENT_TRANS, PROP_LOGIC_EQUIVALENT_REFL);
> val prop_logic_equivalent_preorder =
 PREORDER(('PROP_LOGIC_EQUIVALENT', 'prop_logic'), fn, fn) : preorder

```

## Rewrite Rules

```

> val PROP_LOGIC_EQUIVALENT_rewrites =
[] |- PROP_LOGIC_EQUIVALENT (P_NOT P_TRUE) P_FALSE /\
PROP_LOGIC_EQUIVALENT (P_NOT P_FALSE) P_TRUE /\
...
(!p. PROP_LOGIC_EQUIVALENT (P_AND (p,P_NOT p)) P_FALSE) /\
(!p. PROP_LOGIC_EQUIVALENT (P_AND (P_NOT p,p)) P_FALSE) /\
(!p. PROP_LOGIC_EQUIVALENT (P_OR (p,P_NOT p)) P_TRUE) /\
(!p. PROP_LOGIC_EQUIVALENT (P_OR (P_NOT p,p)) P_TRUE) /\
...
(!p1 p2. PROP_LOGIC_EQUIVALENT (P_AND (p1,P_AND (P_NOT p1,p2))) P_FALSE) /\
... : thm

```

## Congruence Rules

```

> val PROP_LOGIC_EQUIVALENT_congs =
 [] |- (!p1 p1'.
 PROP_LOGIC_EQUIVALENT p1 p1' ==>
 PROP_LOGIC_EQUIVALENT (P_NOT p1) (P_NOT p1')) /\
 ...
 (!p1 p2 p1' p2'.
 PROP_LOGIC_EQUIVALENT p1 p1' ==>
 PROP_LOGIC_EQUIVALENT p2 p2' ==>
 PROP_LOGIC_EQUIVALENT (P_AND (p1,p2)) (P_AND (p1',p2'))) /\
 (!p1 p2 p1' p2'.
 PROP_LOGIC_EQUIVALENT p1 p1' ==>
 PROP_LOGIC_EQUIVALENT p2 p2' ==>
 PROP_LOGIC_EQUIVALENT (P_OR (p1,p2)) (P_OR (p1',p2'))) /\
 ...
 (!p p' s s'.
 PROP_LOGIC_EQUIVALENT p p' ==>
 (s = s') ==>
 (P_SEM s p = P_SEM s' p')) /\
 (!p1 p1'.
 PROP_LOGIC_EQUIVALENT p1 p1' ==>
 (P_IS_TAUTOLOGY p1 = P_IS_TAUTOLOGY p1')) /\
 ... : thm

```

## Creating a Congset

```

- val prop_logic_CS = CSFRAG
 {rewrs = [PROP_LOGIC_EQUIVALENT_rewrites],
 relations = [prop_logic_equivalent_preorder],
 dprocs = [],
 congs = [PROP_LOGIC_EQUIVALENT_congs];}
> val prop_logic_CS =
 CSFRAG{...} : congsetfrag

- val prop_logic_nnf_CS = add_csfrag_rewrites prop_logic_CS
 [PROP_LOGIC_EQUIVALENT_nnf_rewrites];
- val prop_logic_dnf_CS = add_csfrag_rewrites prop_logic_nnf_CS
 [PROP_LOGIC_EQUIVALENT_dnf_rewrites, P_EQUIV_def, P_IMPL_def, P_COND_def];

- val prop_logic_cs = mk_congset [prop_logic_CS];
> val prop_logic_cs = <congset> : congset

- val prop_logic_nnf_cs = mk_congset [prop_logic_nnf_CS];
- val prop_logic_dnf_cs = mk_congset [prop_logic_dnf_CS];

```

## Example: Semantical Equivalence of Propositional Logic

## Usage Examples

```

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_NOT(P_NOT y), x)‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_NOT (P_NOT y),x)) (P_AND (y,x)) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- P_AND (P_PROP (3 + 1),P_PROP (2 + 2)) = P_AND (P_PROP 4,P_PROP 4) : thm

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_PROP (3 + 1),P_PROP (2 + 2))) (P_PROP 4) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_SEM s (P_AND(P_OR(b, b), a))‘‘;
> val it =
 [] |- P_SEM s (P_AND (P_OR (b,b),a)) = P_SEM s (P_AND (b,a)) : thm

```

## Example: Semantical Equivalence of Propositional Logic

## Usage Examples

```

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_NOT(P_NOT y), x)‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_NOT (P_NOT y),x)) (P_AND (y,x)) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- P_AND (P_PROP (3 + 1),P_PROP (2 + 2)) = P_AND (P_PROP 4,P_PROP 4) : thm

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_PROP (3 + 1),P_PROP (2 + 2))) (P_PROP 4) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_SEM s (P_AND(P_OR(b, b), a))‘‘;
> val it =
 [] |- P_SEM s (P_AND (P_OR (b,b),a)) = P_SEM s (P_AND (b,a)) : thm

```

## Example: Semantical Equivalence of Propositional Logic

## Usage Examples

```

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_NOT(P_NOT y), x)‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_NOT (P_NOT y),x)) (P_AND (y,x)) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- P_AND (P_PROP (3 + 1),P_PROP (2 + 2)) = P_AND (P_PROP 4,P_PROP 4) : thm

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_PROP (3 + 1),P_PROP (2 + 2))) (P_PROP 4) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_SEM s (P_AND(P_OR(b, b), a))‘‘;
> val it =
 [] |- P_SEM s (P_AND (P_OR (b,b),a)) = P_SEM s (P_AND (b,a)) : thm

```



## Usage Examples

```

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_NOT(P_NOT y), x)‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_NOT (P_NOT y),x)) (P_AND (y,x)) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- P_AND (P_PROP (3 + 1),P_PROP (2 + 2)) = P_AND (P_PROP 4,P_PROP 4) : thm

- CONGRUENCE_SIMP_CONV ‘‘PROP_LOGIC_EQUIVALENT‘‘ prop_logic_cs std_ss []
 ‘‘P_AND(P_PROP (3+1), P_PROP(2+2))‘‘;
> val it =
 [] |- PROP_LOGIC_EQUIVALENT (P_AND (P_PROP (3 + 1),P_PROP (2 + 2))) (P_PROP 4) : thm

- CONGRUENCE_EQ_SIMP_CONV prop_logic_cs std_ss []
 ‘‘P_SEM s (P_AND(P_OR(b, b), a))‘‘;
> val it =
 [] |- P_SEM s (P_AND (P_OR (b,b),a)) = P_SEM s (P_AND (b,a)) : thm

```

## Problem description

- to handle a PSL formula  $f$  with IBM's model checker RuleBase, it is translated to
  - a satellite automaton  $A$
  - a CTL formula of the form  $AG\ p$
- prove, that this translation is correct for concrete examples, i. e. given  $A, p, f$  prove

$$\forall M. M \parallel A \models_{\text{CTL}} AG\ p \iff M \models_{\text{PSL}} f$$

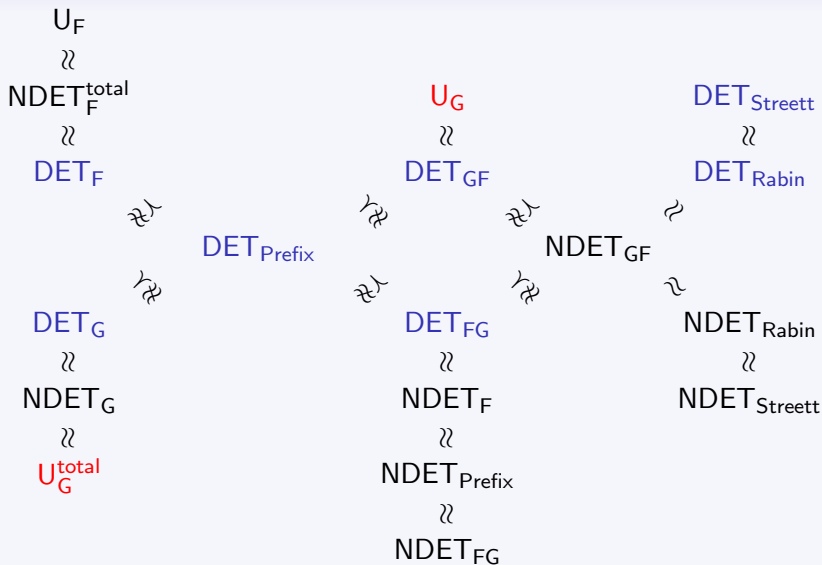
- the combination of  $A$  and  $AG\ p$  can be considered as a universal safety automaton, i. e.

$$\forall M. M \parallel A \models_{\text{CTL}} AG\ p \iff M \models \mathcal{A}_{\forall}(A, G\ p)$$

## Used Method

$$\begin{array}{llll}
 \forall M. M \models \mathcal{A}_\forall(A, G p) & \iff & M \models_{\text{PSL}} f & \text{iff} \\
 \forall M. M \models \mathcal{A}_\forall(A, G p) & \iff & M \models_{\text{LTL}} \text{PSL\_TO\_LTL}(f) & \text{iff} \\
 \forall i. i \models \mathcal{A}_\forall(A, G p) & \iff & i \models_{\text{LTL}} \text{PSL\_TO\_LTL}(f) & \text{iff} \\
 \forall i. i \models \mathcal{A}_{\text{det}}(A_{\text{DET}}, GF p_{\text{DET}}) & \iff & i \models_{\text{LTL}} \text{PSL\_TO\_LTL}(f) & \text{iff} \\
 & & A_{\text{DET}} \models_{\text{LTL}} GF p_{\text{DET}} \leftrightarrow \text{PSL\_TO\_LTL}(f) & 
 \end{array}$$

- $A_{\text{DET}} \models_{\text{LTL}} GF p_{\text{DET}} \leftrightarrow \text{PSL\_TO\_LTL}(f)$  can be handled using the presented tools
- problem: determinisation of  $\mathcal{A}_\forall(A, G p)$



- universal safety automata can express fairness properties
- thus, determinisation of arbitrary universal safety automata results in deterministic Büchi automata
- determinisation of arbitrary universal safety automata needs breakpoints construction
- total universal safety automata are strictly less expressive
- “normal” Rabin-Scott Subset Construction and deterministic safety automata sufficient
- luckily all satellite automata used are total, but the developed tool can also handle non total automata

# Application

- exploiting totality, the developed tool is able to check most examples provided by IBM in less than 5 minutes
- these examples contain some non toy examples
- IBM is interested in using this tool for verifying the current method and for debugging new developed features
- a bug in IBM's translation has been found

# Conclusions

- deep embeddings of important temporal logics provided
- fully automatic translation of *LTL* and a significant subset of *PSL* to  $\omega$ -automata
- model checking of these formalisms using *SMV* possible
- side effect: additional interface to the HOL simplifier, that might be useful in general
- these deep embeddings and translations can be used for an application, IBM is interested in

## Possible Future Work

- do more examples with the developed tool
- create an input parser
- implement some details more efficiently
- extend the subset of *PSL* that can be translated to full *PSL*