I C what you did last summer
2016

A robotika sok törvénye

:: Gazder Bence “Gazben” - 5. év Msc

Open Source Robotics Foundation

Mit szeretnénk elérni?

- Legyen egy szabály leíró nyelv
- Írjunk le szabályokat
- Ezt valahogy magyarázzuk el a robotnak
- Amennyiben a szabály sérül működés közben akkor a robot szóljon nekünk

DEMO

WoW - Mi is történt az előbb?

- Remélhetőleg megtörtént amit vártunk
  - A beadott parancsot értelmezte parser
  - Az értelmezésből gyártott egy forráskódot
  - Mi a forráskódot lefordítottuk
  - A lefordított program megfigyelte a rendszer egyes csatornáit és eldöntötte, hogy teljesültek-e feltételek

ROS - honnan tudjuk, hogy mit történik?

- A neve ellenére nem operációs rendszer, “csak” egy platform
- Folyamatok közötti kommunikációt segíti elő
- localhoston keresztül kommunikál
- Ezt a kommunikációt “le lehet hallgatni”, tehát bármelyik másik modul tudhatja, hogy mi történik a rendszer egy másik részén

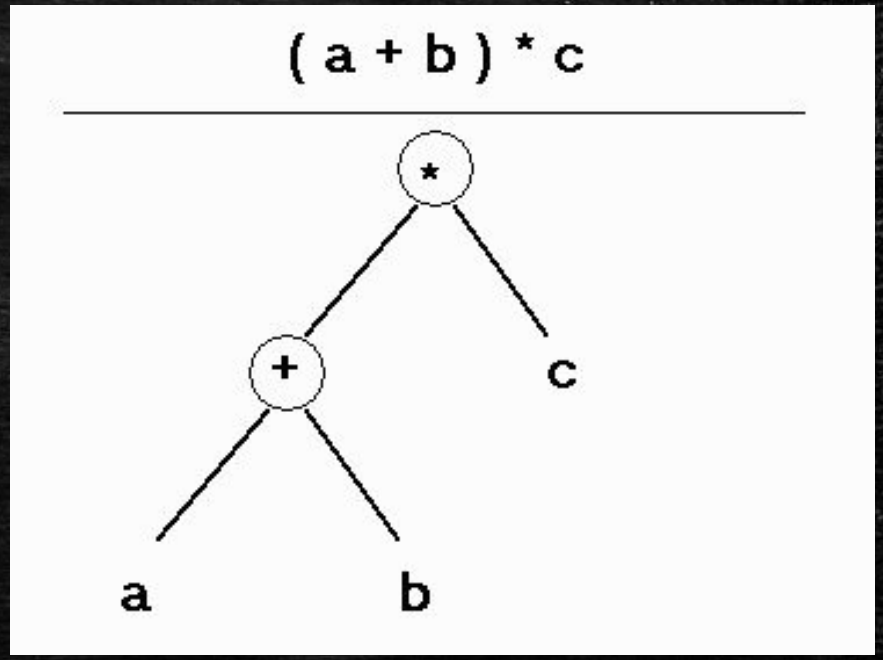
LTL - a szabályleíró

- Gyakorlatilag a digitből bool algebra kiterjesztése
- Egymás utáni események közötti logikai kapcsolatot vizsgál
- Ez azért jó, mert pontosan ez történik a valóságban
- Példa kifejezés: G(r -> (pUd))

SyntX - a szabályértelmező

Forráskód: https://gitlab.com/nagygr/syntx

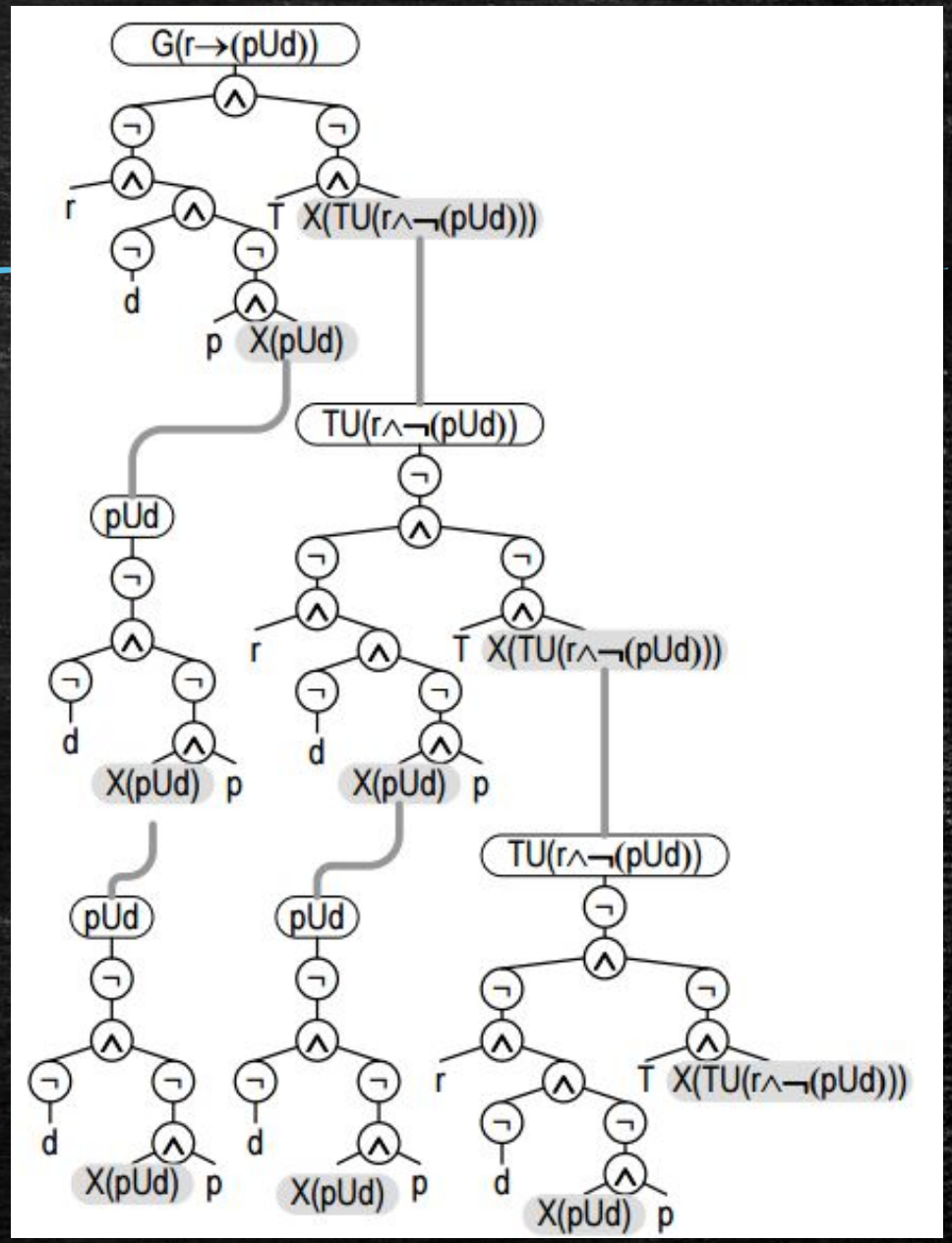
- Cpp11 tárgyból tanultuk. Egyszerűbben használhatót nem találtam.
- EBNF-et használ a nyelvek leírására
- Abstrakt Szintaxis Fát (AST) épít



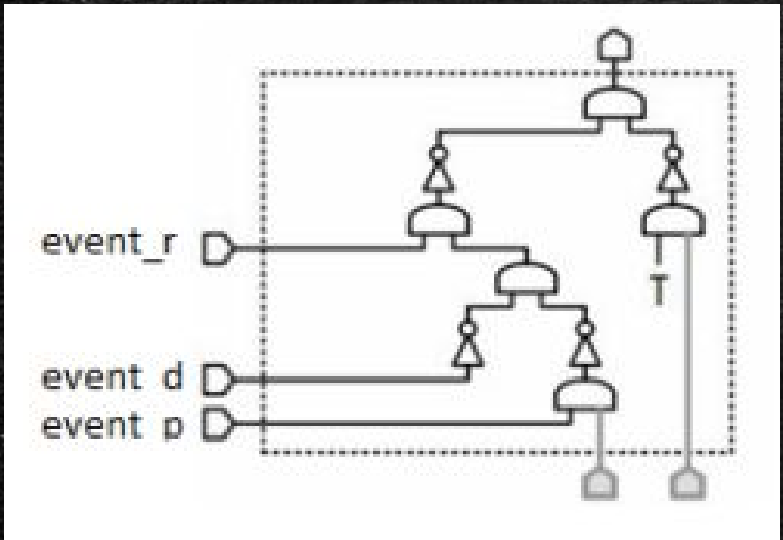
LTL -> AST -> Blokk

Az LTL nyelv definiál bizonyos ekvivalenciákat. Ebből nekünk a legfontosabb:

x U y ekvivalens y ∨ (x ∧ X(x U y))



Monitor - Mit szeretnénk csinálni?



- Egy oda vissza láncolt listát ahol a láncelemek egy kiértékelő blokknak felenek meg
- Kimenetek és bemenetek trilean típusúak

```
class Property {
protected:
  static Property* current_block;
  /* .... */
  unsigned int id;
  StateRegister* state_register_ptr;
public:
  std::vector<trilean> input_states;
  std::vector<trilean> output_states;
  Property* root_node;
  Property* children_node;
  std::function <class Property*(class Property*)>
construct_children_func;
  std::vector <std::function<trilean(class Property*)>>
eval_functions;
  Property* constructChildrenBlock();
  trilean isEventFired(StateRegisterType);
  trilean evaluate();
};
```

Generálás - süssük ki amit főztünk

- Be kell járni a korábban elkészített AST-t
- Le kell másolni a megadott mappába a fájlokat
- Ki kell cserélni az előre definiált szövegeket a generálásnak megfelelőre pl.: //--EVENTS--

Generálás - események

Előtte

```
#ifndef Events_h__
#define Events_h__

typedef unsigned long long int
  StateRegisterType;

//dedicated end event
const StateRegisterType EVENT_END = 1;

//--EVENTS--

#endif // Events_h__
```

Utána

```
#ifndef Events_h__
#define Events_h__

typedef unsigned long long int StateRegisterType;

const StateRegisterType EVENT_END = 1;
const StateRegisterType event_d = 2;
const StateRegisterType event_p = 4;
const StateRegisterType event_r = 8;

#endif // Events_h__
```

Generálás - egy modul

```
inline trilean
 EVAL_AndNotAndevent_rAndNotrevent_dNotAndevent_pNotAnd(Property
 * property);

inline Property* construct_block0(Property* root_node);
inline Property* construct_block0(Property* root_node){
root_node->eval_functions.push_back(EVAL_AndNotAndevent_rAndNotreve
 nt_dNotAndevent_pNotAnd);
 root_node->construct_children_node_func = construct_block1;
 root_node->output_states.resize(1);
 root_node->input_states.resize(2);
 return root_node;
}
```

:: ROS /
MONITOR

Forráskód: https://github.com/gazben/monitor_generator

Mottó: ha valami bonyolultat akarsz csinálni, akkor ez legyen nagyon bonyolult

Köszönöm a figyelmet