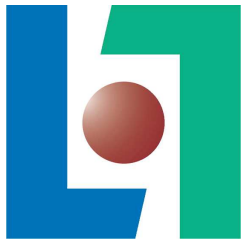

$\ell A\ell B\ell C$: a program testing the equivalence of dpda's

P. Henry, G. Sénizergues

Bordeaux 1 university, LaBRI,

January 29-th 2013

LSV seminar

contents

The PROBLEM

Instance: A, B deterministic pushdown automata.
Question: $\mathrm{L}(A) = \mathrm{L}(B)$?

Let us Compute.

Automata,grammars

pushdown automata $\leftrightarrow$ context-free grammars
Deterministic pushdown automata $\leftrightarrow$ strict-deterministic c.f. grammars [Harrison-Havel JCSS 1993]

Theorem

The equivalence problem for dpda is decidable.

[GS ICALP 1997] [GS TCS 2001, page 80]
The problem is primitive recursive.
[C. Stirling ICALP 2002]

The program: WHAT ?

Let $G =< X, V, P >$ a dcf grammar.
Let $W_1, W_2 \in (X \cup V)^*$. We call a witness of non-equivalence every terminal word $w \in X^*$ such that

$$w \in (L(W_1) - L(W_2)) \cup (L(W_2) - L(W_1)).$$

Example 0:
$G =< \{a, b, c\}, \{S, T\}, P >$ where P consists of the rules:

$$\begin{aligned} S &\rightarrow aSb + c \\ T &\rightarrow aTb + c \end{aligned}$$

$S \equiv T$ can be proved by the following G-equality-proof

$$\dfrac{\dfrac{S \equiv aSb + c}{S \equiv T}\,gr \quad \dfrac{\dfrac{\dfrac{\dfrac{}{a \equiv a}\,ref \quad \dfrac{}{Sb \equiv Tb}\,sj}{aSb \equiv aTb}\times \quad \dfrac{}{c \equiv c}\,ref}{aSb + c \equiv aTb + c}+ \quad \dfrac{T \equiv aTb + c}{aTb + c \equiv T}\,gr}{aSb + c \equiv T}\,sym}{S \equiv T}\,trans \quad \dfrac{}{b \equiv b}\,ref}{Sb \equiv Tb}\,trans\times$$

Example 0: Let $P := \{(S, T)\}$
This set P is self-provable: $\forall (W, W') \in P, \forall x \in \{a, b, c\}$

$$(W \odot x, W' \odot x)) \in \mathrm{CC}(P).$$

$$\begin{aligned} (S \odot a, T \odot a) &= (Sb, Tb) \\ (S \odot b, T \odot b) &= (0, 0) \\ (S \odot c, T \odot c) &= (\varepsilon, \varepsilon) \end{aligned}$$

$$(Sb, Tb) \in \mathrm{CC}(\{(S, T)\})$$

G-EQuality (schemes of) rules:

$$\frac{x \equiv y}{y \equiv x}\, sym \qquad \frac{x \equiv y \;\; y \equiv z}{x \equiv z}\, trans \qquad \frac{x \equiv x' \;\; y \equiv y'}{x + y \equiv x' + y'}\, +$$

G-EQuality (schemes of) strict-rules:

$$\frac{}{x \equiv x}\, ref \qquad \frac{x \equiv x' \;\; y \equiv y'}{x \cdot y \equiv x' \cdot y'}\, \times$$

G-EQuality strict-rules:

$$\frac{}{S \equiv \sum_{i=1}^{n} w_i}\, gr$$

Instanciations, for sym,trans,+,ref,$\times$: by rational languages.
Instanciations, for gr: exactly one instance for each grammar rule.

Definition

Let G be a cf grammar. We call G-equality proof, every finite set $\mathcal{J} \subset \mathrm{RAT}((X \cup V)^*)$ such that: $\mathcal{J}$ decomposes as $\mathcal{J} = P \cup Q$ with
1- Every jydgment $q \in Q$ has a proof within the system G-EQ, with hypotheses in P.
2- Every judgment $p \in P$ is a consequence, by some strict rule of G-EQ of some judgments in Q.

Remark:
From a self-provable set P and the sequence of rules establishing that $(W \odot x, W' \odot x) \in \mathrm{CC}(P)$, one can construct, in linear time, a G-equality-proof for all the pairs of P.

INPUT:
A strict det grammar and two words W_1, W_2.
OUTPUT:
- either a witness of $W_1 \not\equiv W_2$
- or a self-provable set posessing (W_1, W_2)

The program: HOW ?

Theorem

1- For every cf grammar G, the system G-EQ is sound for the equality of the generated languages.
2- For every strict deterministic cf grammar G, the system G-EQ is complete for the equality of the generated languages.

Proof:
Completeness of D_0 [GS TCS 2001,p.80]
Elimination-lemma [GS TCS 2001,p.100] $\rightarrow$ completeness of D_5
This D_5-proof $\rightarrow$ a self-provable set $\mathcal{J}$

First tentative:
- enumerate words
- enumerate equality proofs in G-EQ

in parallel.

Second tentative:
Compute N=f($\|A\| + \|B\|$).
- enumerate words (up to length N)
- or enumerate proofs in G-EQ (up to depth N).

Better idea:
extract from the completeness proof (for D_0) some algebraic/algorithmic ideas leading to:
- either a witness of falsity
- or a proof

X: alphabet. Semi-groupoid of prefix matrices.
An element of $\mathrm{D}\mathbb{B}_{p,q}\langle\langle\ X\ \rangle\rangle$ is a p-q matrix with coefficients in $\mathbb{B}\langle\langle\ X\ \rangle\rangle$, such that every line is *prefix*.

$$W \bullet x := (x)^{-1} W$$

$G =< X, V, P >$: strict deterministic grammar.
An element of $\mathrm{D}\mathbb{B}_{p,q}\langle\langle\ V\ \rangle\rangle$ is a p-q matrix with coefficients in $\mathbb{B}\langle\langle\ V\ \rangle\rangle$, such that every line is *deterministic* (notion inherited from the grammar).

$$v \odot x := \sum_{v \to m} m \bullet x$$

The map:

$$M = (m_{i,j}) \mapsto (\mathrm{L}(m_{i,j}))$$

- is compatible with matrix product
- is equivariant for the right-operations $\bullet$ and $\odot$.

X: alphabet. Let $\vec{\alpha}, \vec{\beta} \in \mathrm{D}\mathbb{B}_{1,q}\langle\langle\ X\ \rangle\rangle$. A unifier of $(\vec{\alpha}, \vec{\beta})$ is any matrix $U \in \mathrm{D}\mathbb{B}_{q,q}\langle\langle\ X\ \rangle\rangle$ such that:

$$\vec{\alpha} \cdot U = \vec{\beta} \cdot U.$$

U is a MGU iff, every unifier has the form $U \cdot T$ with $T \in \mathrm{D}\mathbb{B}_{q,q}$. This notion is lifted to $\vec{\alpha}, \vec{\beta} \in \mathrm{DR}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$

Theorem

1- Every pair $\vec{\alpha}, \vec{\beta} \in \mathsf{D}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ *has a MGU (up to* $\equiv$*)*
2- This MGU is unique, up to $\equiv$ *and up to some right-product by a permutation matrix.*
3- For pairs $\vec{\alpha}, \vec{\beta} \in \mathsf{DR}\mathbb{B}_{1,q}\langle\langle\ V\ \rangle\rangle$ *the MGU has some representative which is in* $\mathsf{DR}\mathbb{B}_{q,q}\langle\langle\ V\ \rangle\rangle$ *and is computable from* $\vec{\alpha}, \vec{\beta}$.

Logical/deduction rules

Every logical rule:

$$J1, J2, \ldots, Jn \rightarrow J'$$

has the property that:

$$\sup\{\nu(J1), \nu(J2) \ldots, \nu(Jn)\} \leq \nu(J')$$

(truth value increases)
Every deduction rule:

$$J1, J2, \ldots, Jn \vdash\!\!- J'$$

has the property that:

$$\sup\{H(J1), H(J2) \ldots, H(Jn)\} \leq H(J')$$

(cost-function H increases)

For the judgment:

$$J = (n, U, V)$$

with n integer, $U, V \in \mathrm{D}\mathbb{B}_{1,2}\langle\langle\ V\ \rangle\rangle$.

$$H(J) := n + \inf\{|w| \mid w \in X^*, w \in \mathrm{L}(U) \Leftrightarrow w \notin \mathrm{L}(V)\}$$

Typical logical rule:

$$(p, U, V) \to (p + 1, U \odot x, V \odot x)$$

Typical deduction rule:

$$\{(p + 1, U \odot x, V \odot x) \mid x \in X\} \mid\!\!\vdash\!\!- (p, U, V)$$

Comparison trees

Starting from a node we build a comparison-tree:
- a node is closed or open
- every node is a logical-consequence of the root
- every " closed" node is a deduction-consequence of other nodes (usually its sons, but not always)
- when the comparison-tree has closed nodes only, it is a self-provable set.

A tactics:
is applied to one node n.
It modifies the comparison-tree, locally, around this node.

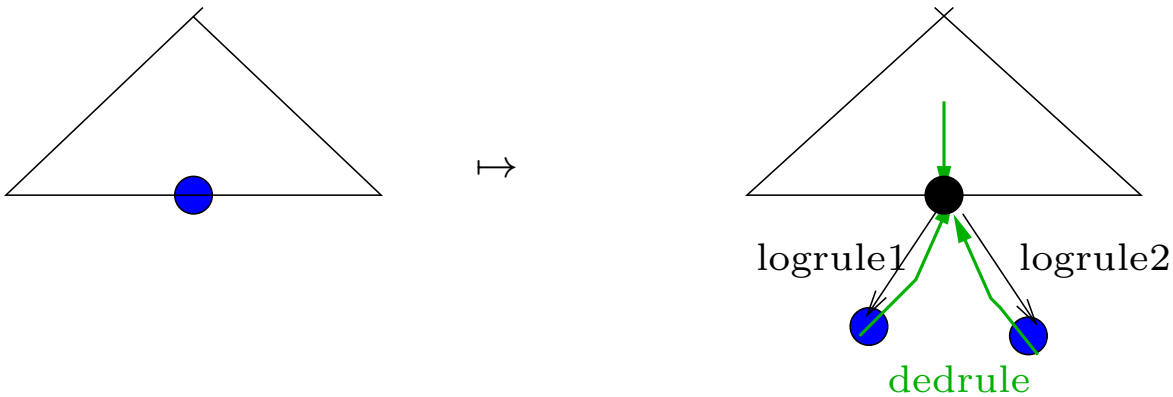

Figure : a tactics

Simple tactics

TA:

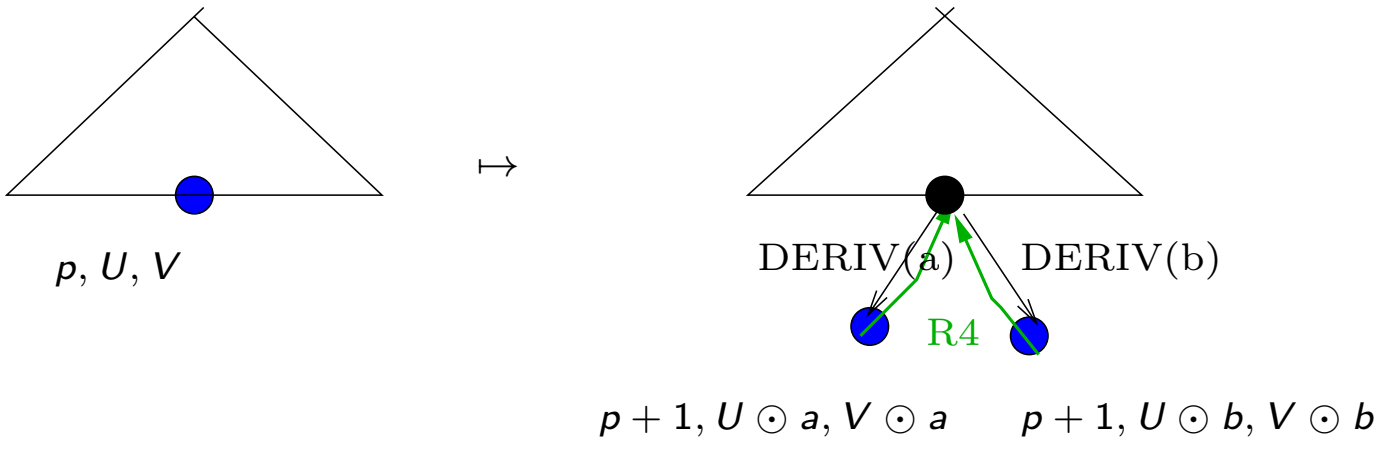

Figure : TA

Simple tactics

TD:

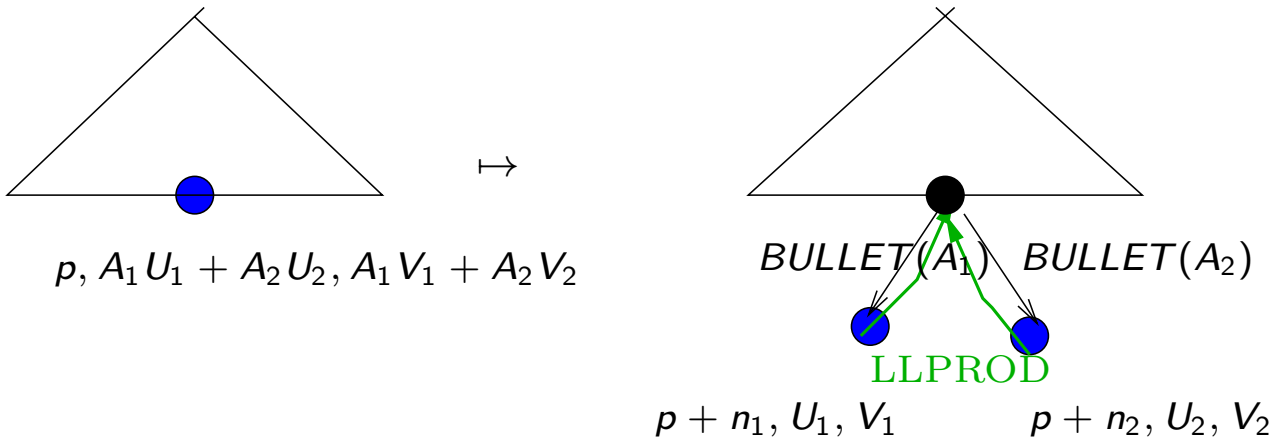

Figure : TD

Simple tactics

Teq:

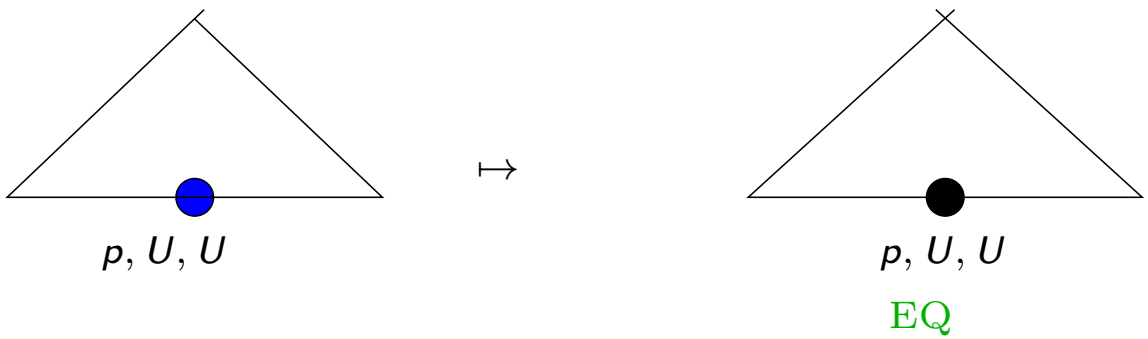

Figure : Teq

Simple tactics

TREP:

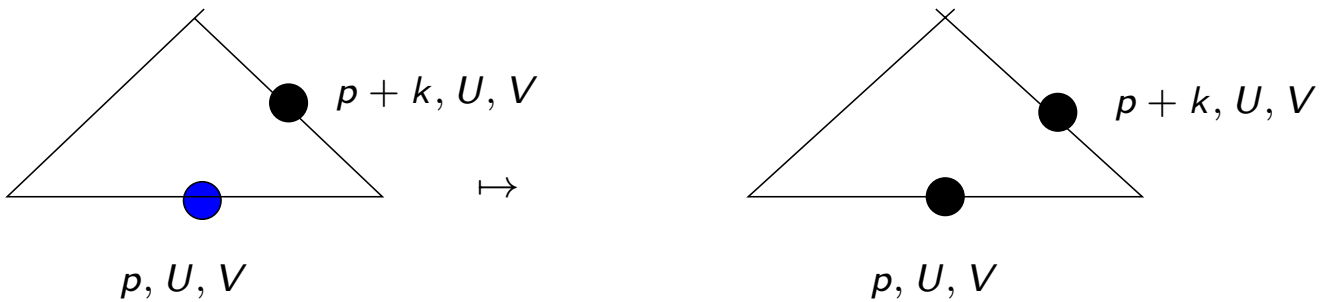

Figure : TREP

Triangulation

TCM ([GS,TCS 2001 pages 50-60]):

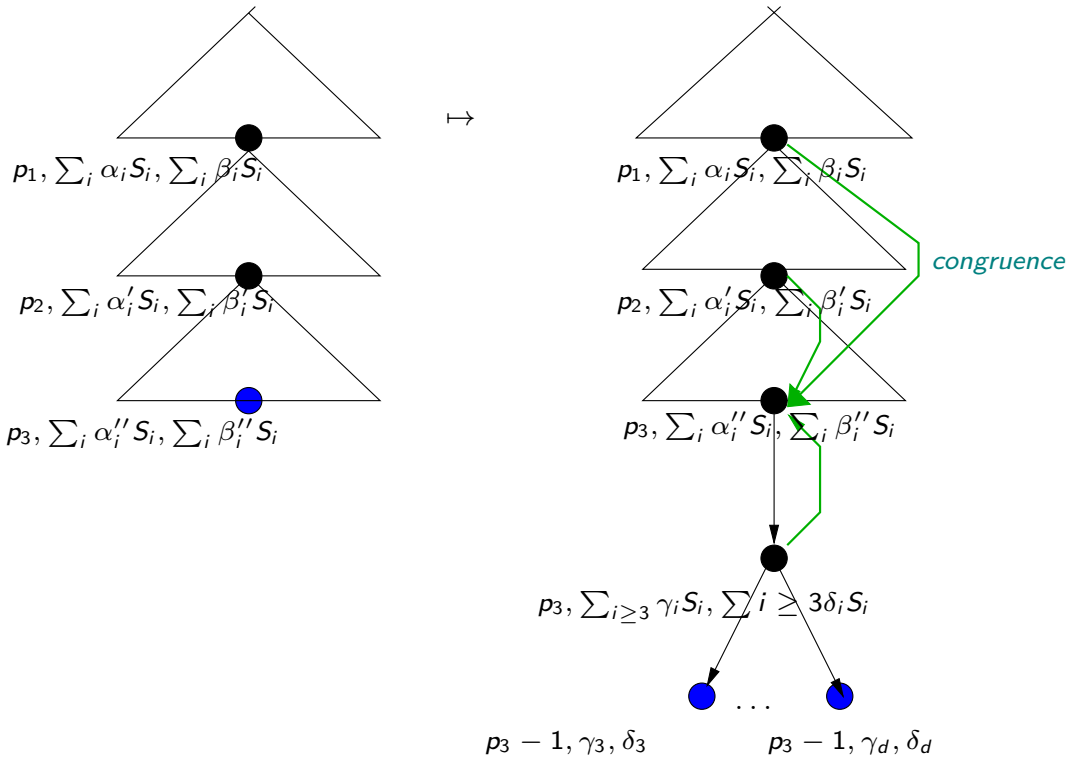

Figure : TCM

Dynamic Unifiers

TCR:

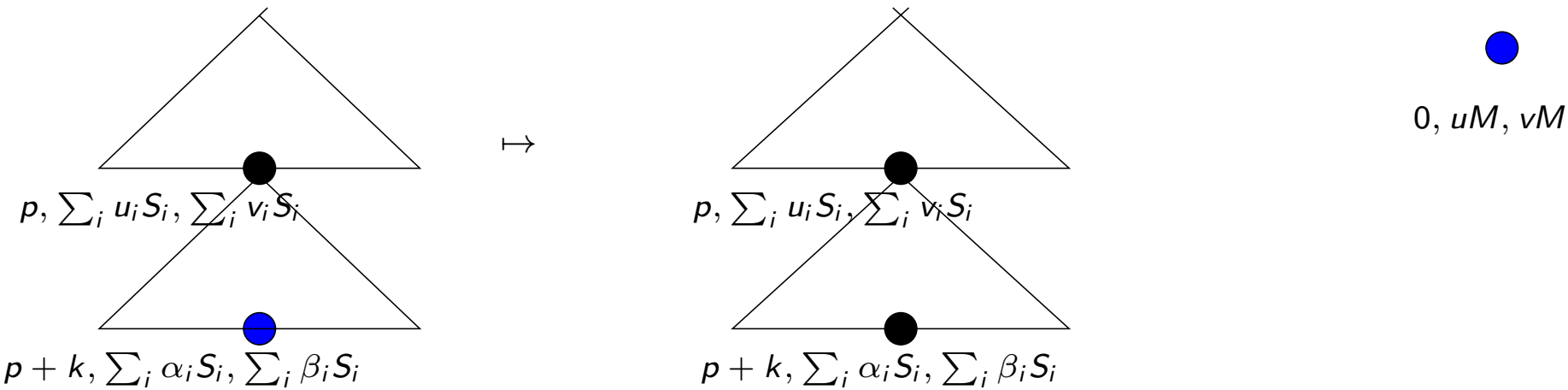

Figure : TCR

witness-lift: function that " lifts" the witness of falsity (initially ε) from a leaf up to the root or to a unifier-computation.

A strategy is built from a sequence of tactics and a priority ordering over these tactics.
Implemented by a functionnal:

```
def make-strategy(maxsteps,error-tactics,*tactics)
```

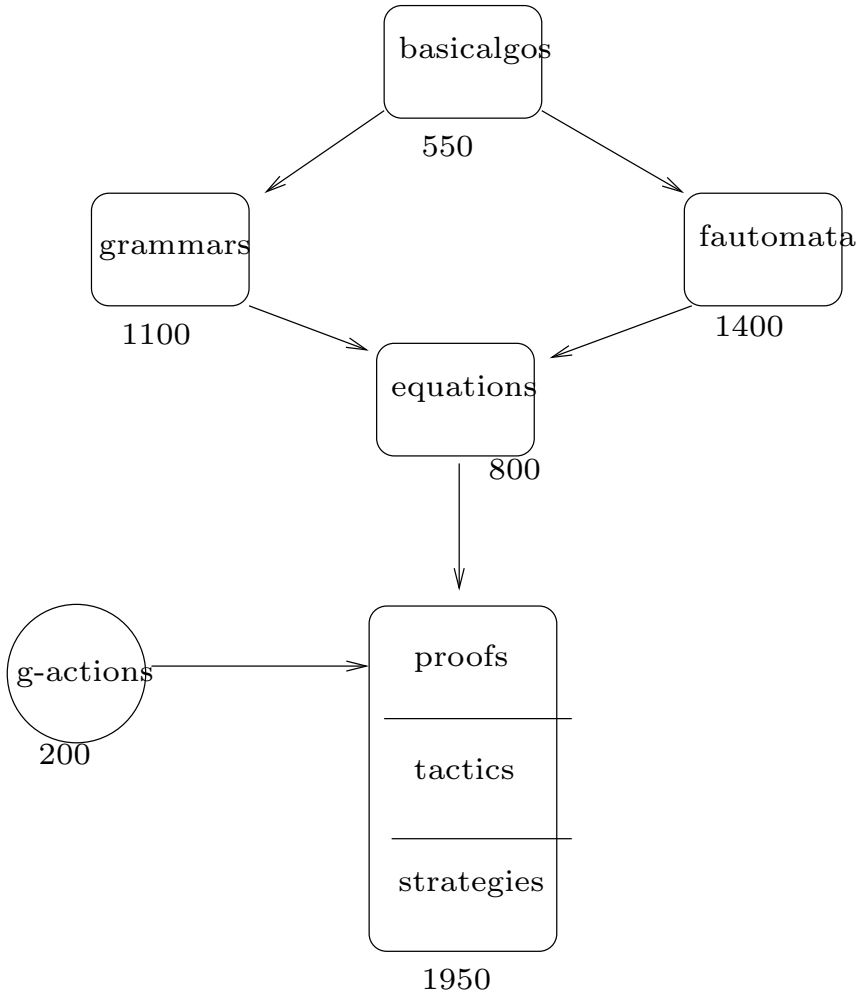

Figure : structure of the program

The program:

EXAMPLES

Input: example2
A2=dpda:

$$
\begin{aligned}
q1, O, \# &\rightarrow q1, A, O \\
q2, O, \# &\rightarrow q2, A, O \\
q1, A, a &\rightarrow q3 \\
q3, A, a &\rightarrow q3 \\
q3, O, a &\rightarrow q3p, O \\
q1, A, b &\rightarrow q5 \\
q5, A, \varepsilon, &\rightarrow q5 \\
q3p, O, a &\rightarrow q3b, O \\
q1, A, x &\rightarrow q1, A, A \\
q5, O, \varepsilon &\rightarrow q3
\end{aligned}
$$

$$
\begin{aligned}
q3b, O, a &\rightarrow q3 \\
q2, A, a &\rightarrow q4, A, A \\
qb, O, \varepsilon &\rightarrow q3 \\
q4, O, a &\rightarrow q3 \\
q2, A, b &\rightarrow qb \\
q2, A, x &\rightarrow q2, A, A \\
qb, A, \varepsilon &\rightarrow qb \\
q4, A, a &\rightarrow q4
\end{aligned}
$$

$$< q1 - O - q3 > \equiv < q2 - O - q3 > ?$$

---

20 is the depth used for computing the triangulations
replacing "infty" by an integer is suitable for cautious experiments

---

```
TC=(lambda P,n:TCM(P,n,20))
strategy-DCMA=
make-strategy("infty",Terror,Teq,Trep,TD,TC,TA)
```

Output:136 nodes, 2 TC

Input: example2

---

20 is the depth used for computing the qmgu's
the mgu are stored in a dictionnary (and thus computed only once)
only one jump is made after the mgu-computation

```
TC=(lambda P,n:TCJ-qmgu1-dico(P,n,20))
strategy-DCJqmgudA=
make-strategy("infty",Terror,Teq,Trep,TD,TC,TA)
```

Output: 109 nodes, 2 TC

Input:example2

---

10 is the depth used for computing the qmgu's
the unifiers are stored in a dictionnary;
unifiers are tested at root positions
errors are used for reaching the mgu
only one jump is made after the mgu-computation; tree is stopped after the jump

---

```
TC=(lambda P,n:TCR-qmgu1(P,n,10))
strategy-DCRqmgudynA=
make-strategy(50000,Terror-dyn,Teq,Trep,TD,TC,TA)
```

Output:61 nodes, 2 unifiers

Input:example2

---

10 is the depth used for computing the qmgu's
the unifiers are stored in a dictionnary;
unifiers are tested at root positions
TCJ is tried first, TSUN is a second choice
error detection causes failure of the strategy

```
TC=(lambda P,n:TCR-qmgu1-static(P,n,10))
TS=(lambda P,n:TSUN-qmgu1-static(P,n,10))
strategy-DCSRqmgustatA=
make-strategy(50000,Terror,Teq,Trep,TD,TS,TC,TA)
```

Output: 54 nodes, 3 unifiers

Input:example32 (108 grammar rules)

```
TC=(lambda P,n:TCR-qmgu1-static(P,n,10))
TS=(lambda P,n:TSUN-qmgu1-static(P,n,10))
strategy-DCSRqmgustatA=
make-strategy(50000,Terror,Teq,Trep,TD,TS,TC,TA)
```

Output: 502 nodes, 9 unifiers

PERSPECTIVES

Translation:
comparison-tree $\rightarrow$ self-proved set of equations $\rightarrow$ G-equality-proof
principle:follow the induction from [GS, TCS 2001 pages 80-100]
price to pay: size of the equality-proof $>$ size of the comparison-tree.
advantage: such a proof is convincing:
can be tested by easy-to-understand and easy-to-write programs.

use term rewriting $\rightarrow$ simplify equations
use polynomial rewriting $\rightarrow$ test that an equation is consequence of others.

one-counter, without ϵ-transitions [Göller ICALP 2011]
one-turn dpda, finite-turn dpda [GS ICALP 2003]
finite languages.

First-order program-schemes
First-order grammars
Equational graphs: isomorphism problem, quotient problem
Deterministic finite transducers
General overview [GS MCU 2001]
Cryptographic protocols [Chretien-Cortier-Delaune in progress]

Grammar transformations
Bi-compression of words, trees
Composition of examples.

Right-now:$X^* \to \{0,1\}$
Maps: $X^* \to \mathbb{Z}$ [GS TCS 2001, pages 100-160]
Maps: $X^* \to F(Y)$?[GS ICALP 1999]
Bisimulation of non-det pda with deterministic decreasing ϵ-moves ([GS SIAM 2005]) ??