Improving Automated Feedback

Building a Rule Feedback Generator

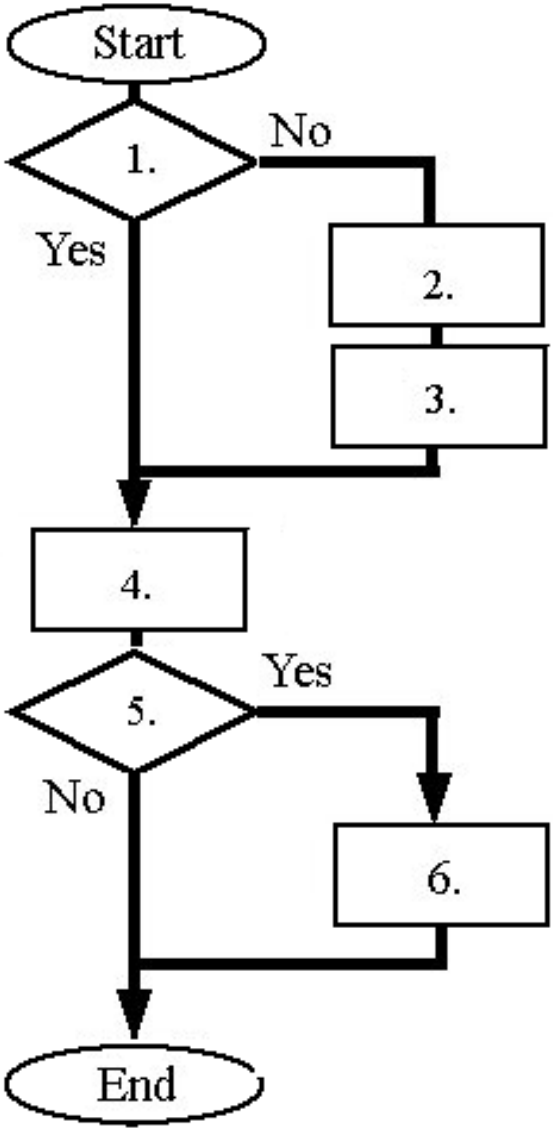
Eric Bouwers

September 27, 2007

(Generic) Outline

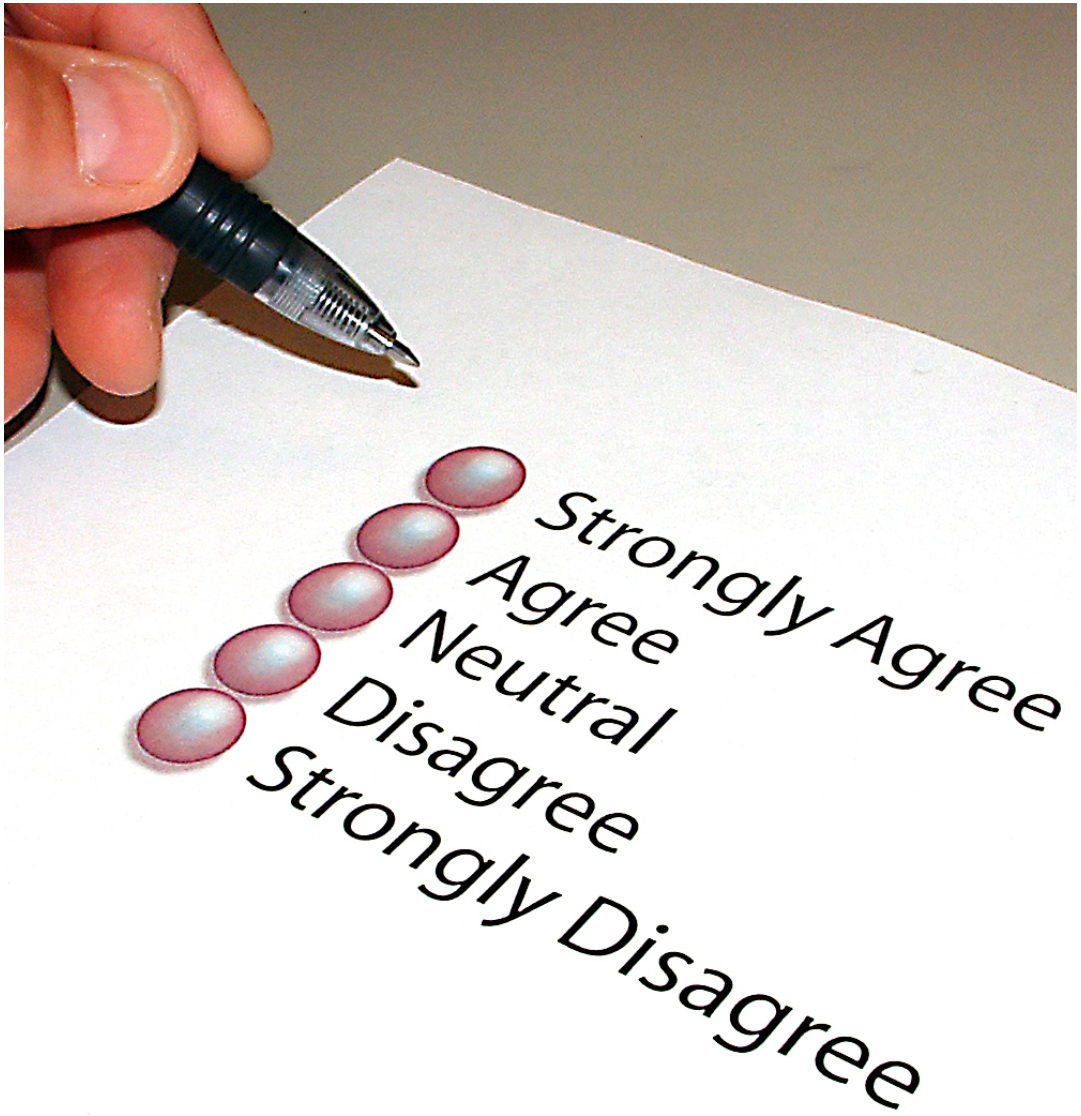
1. The problem
2. Our solution
3. Evaluation
4. Conclusion

Procedural skills



1. Are the denominators the same? If yes, skip to step 4.
2. Find the lowest common denominator.
3. Convert both fractions to the common denominator.
4. Add the numerators, and place the sum over the common denominator.
5. Can the answer be simplified? (If no, you are finished.)
6. Simplify the answer.

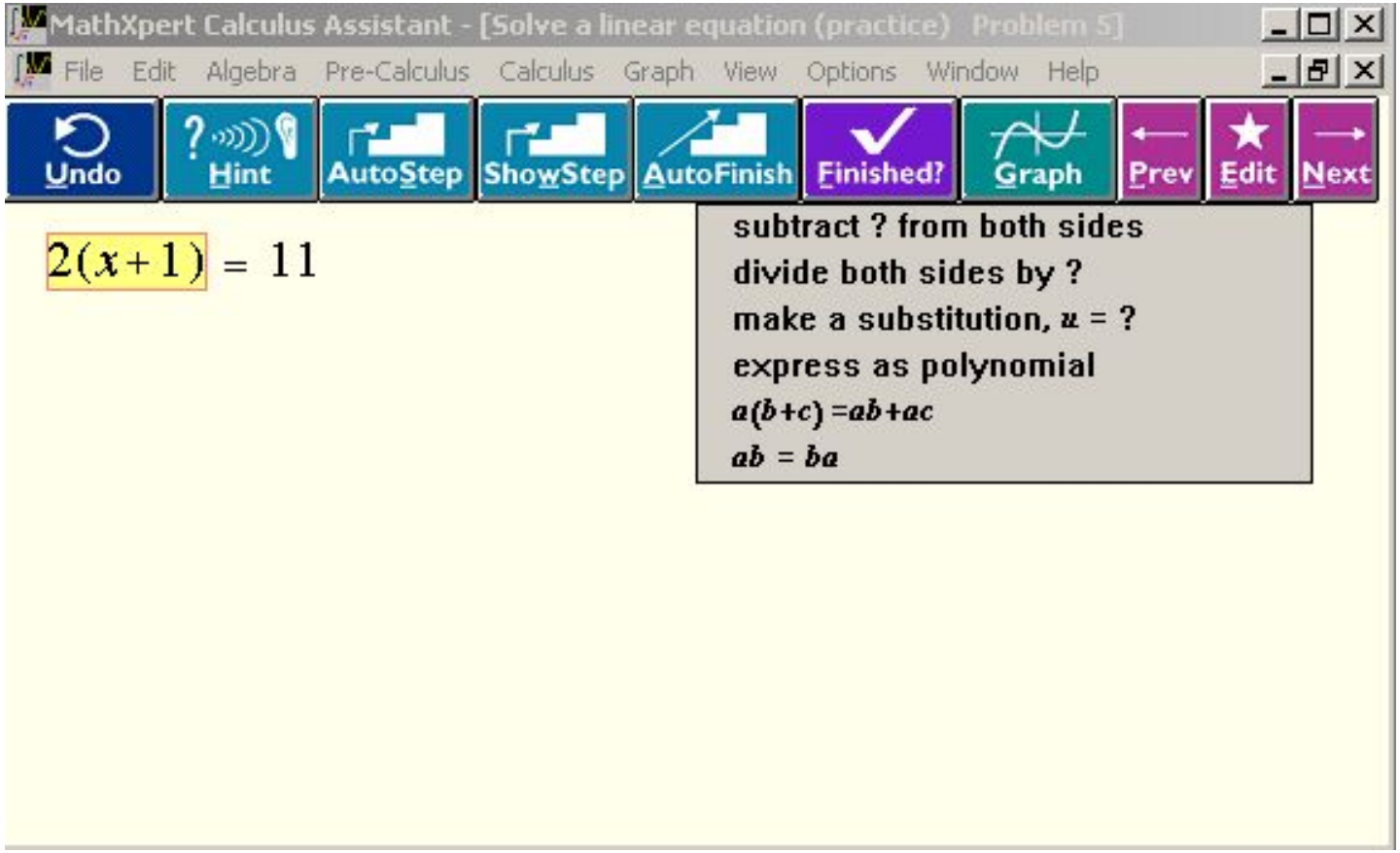
Feedback



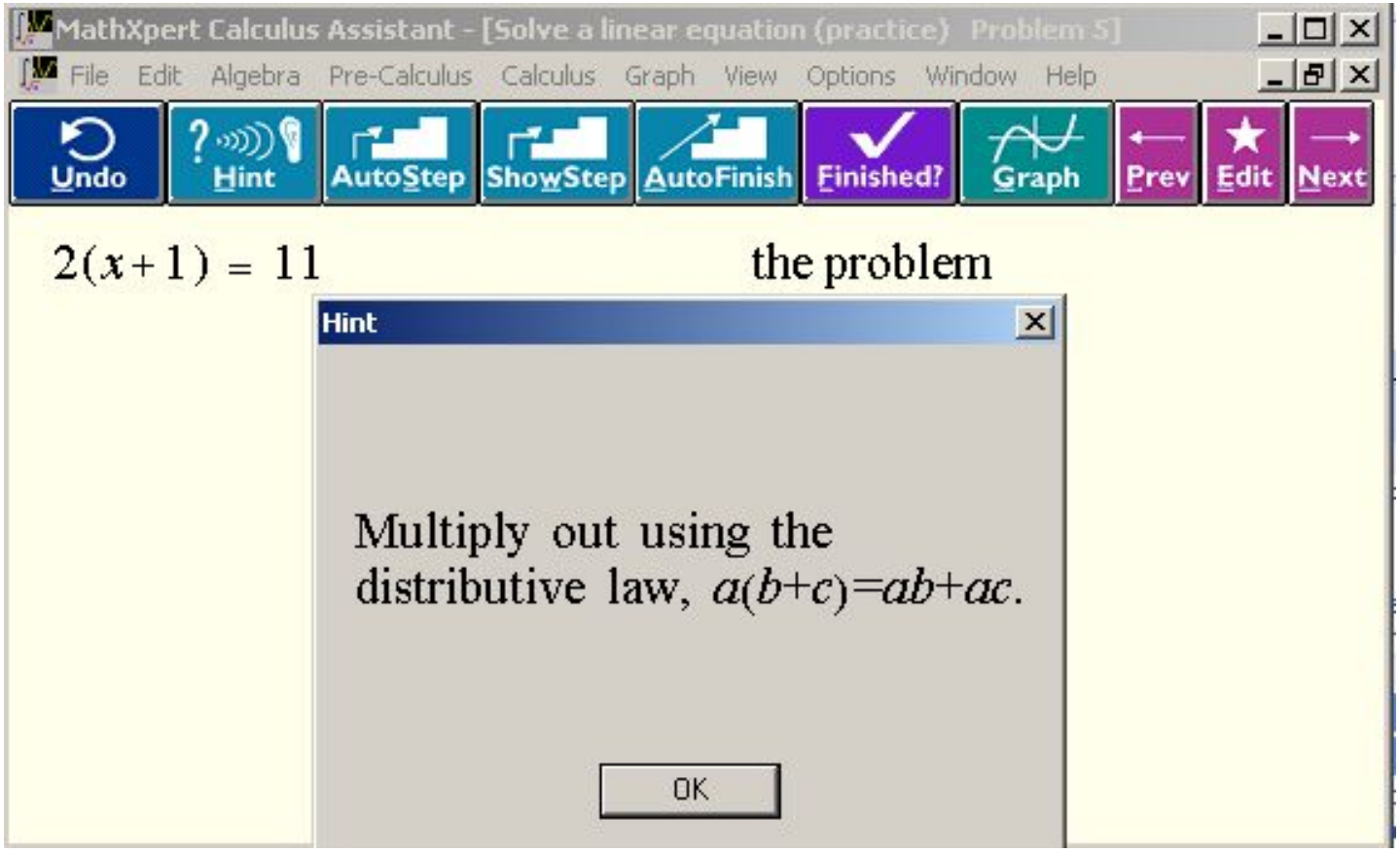
Feedback in Education

Feedback in Education

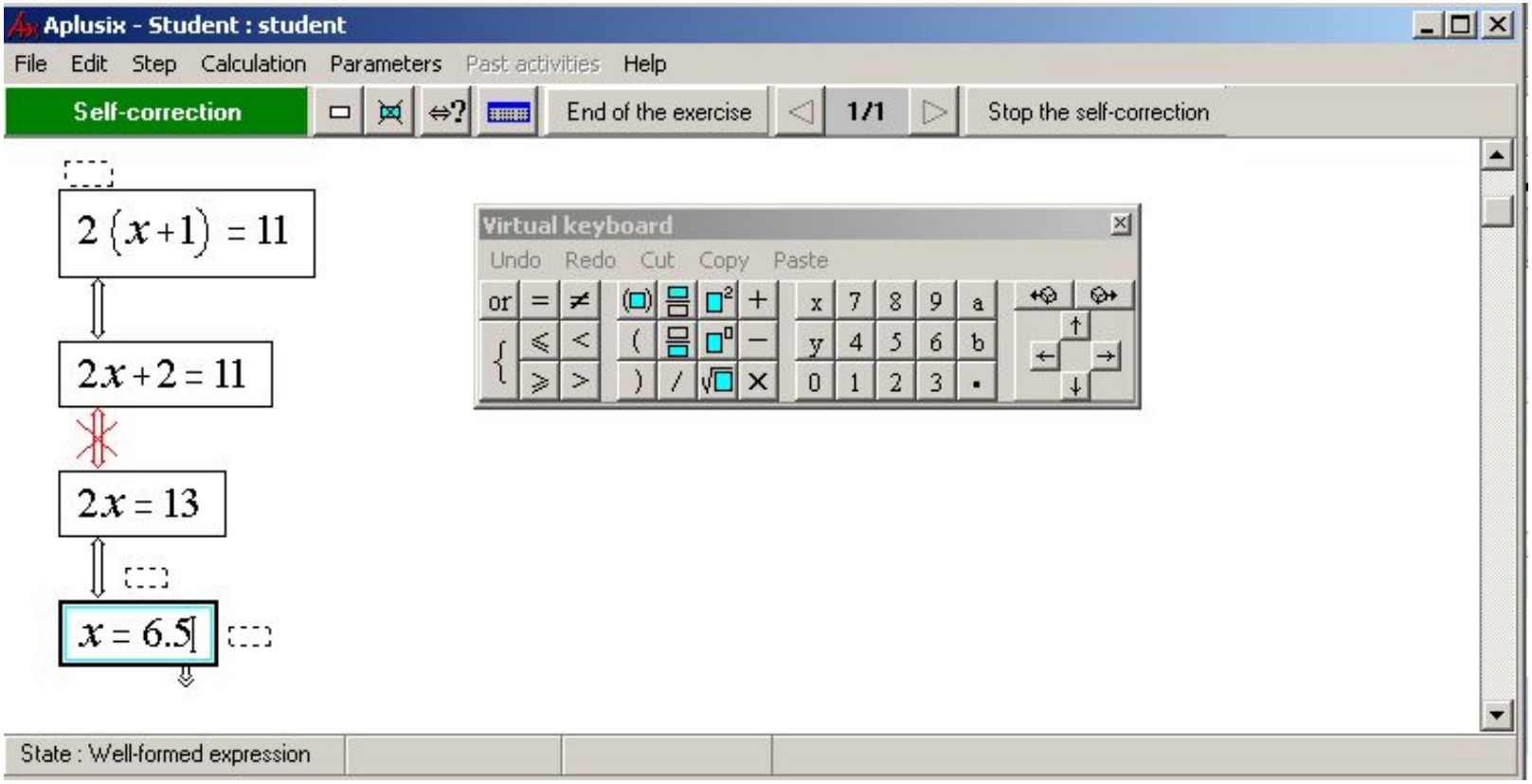
Current solutions



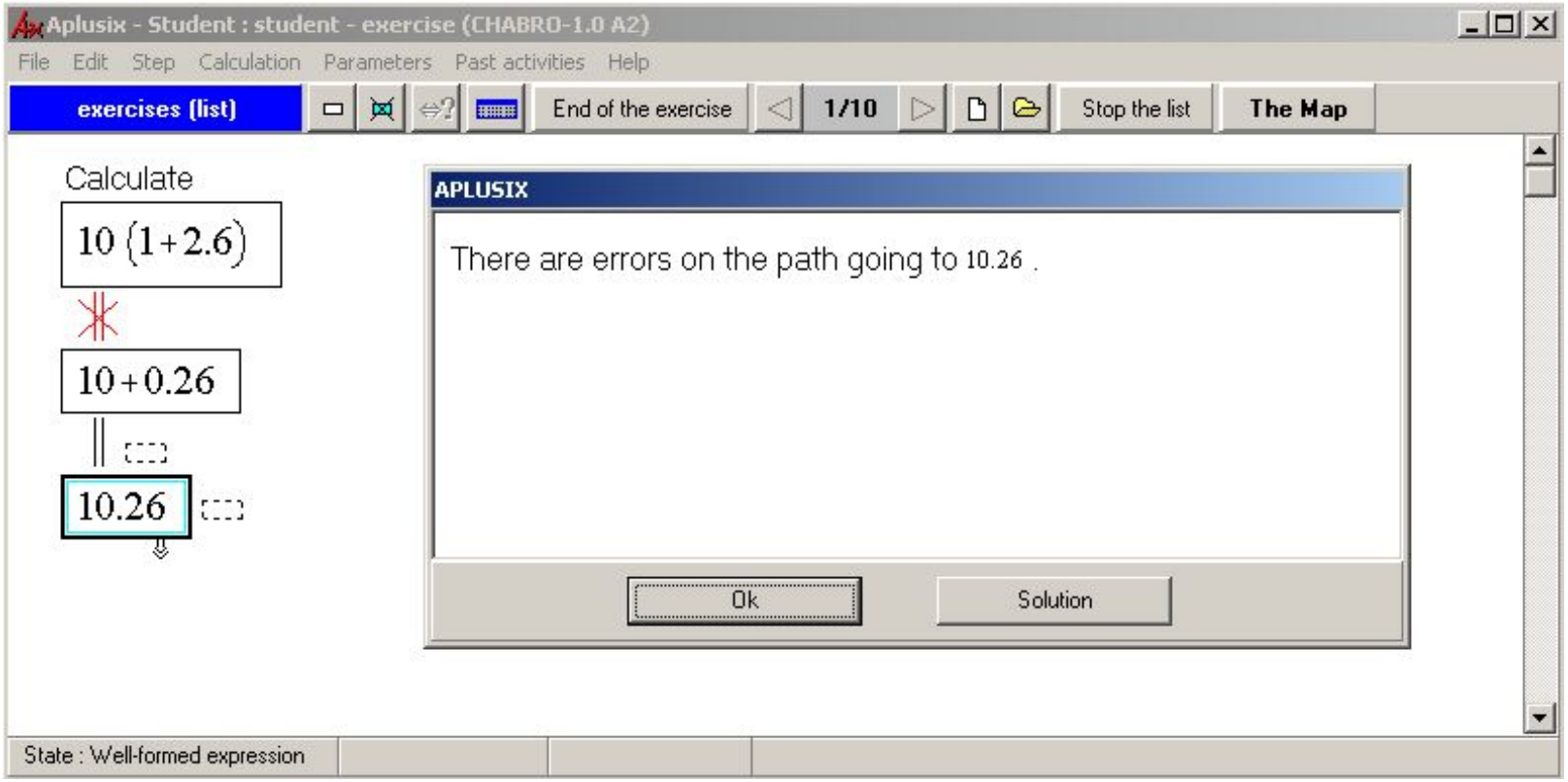
Current solutions



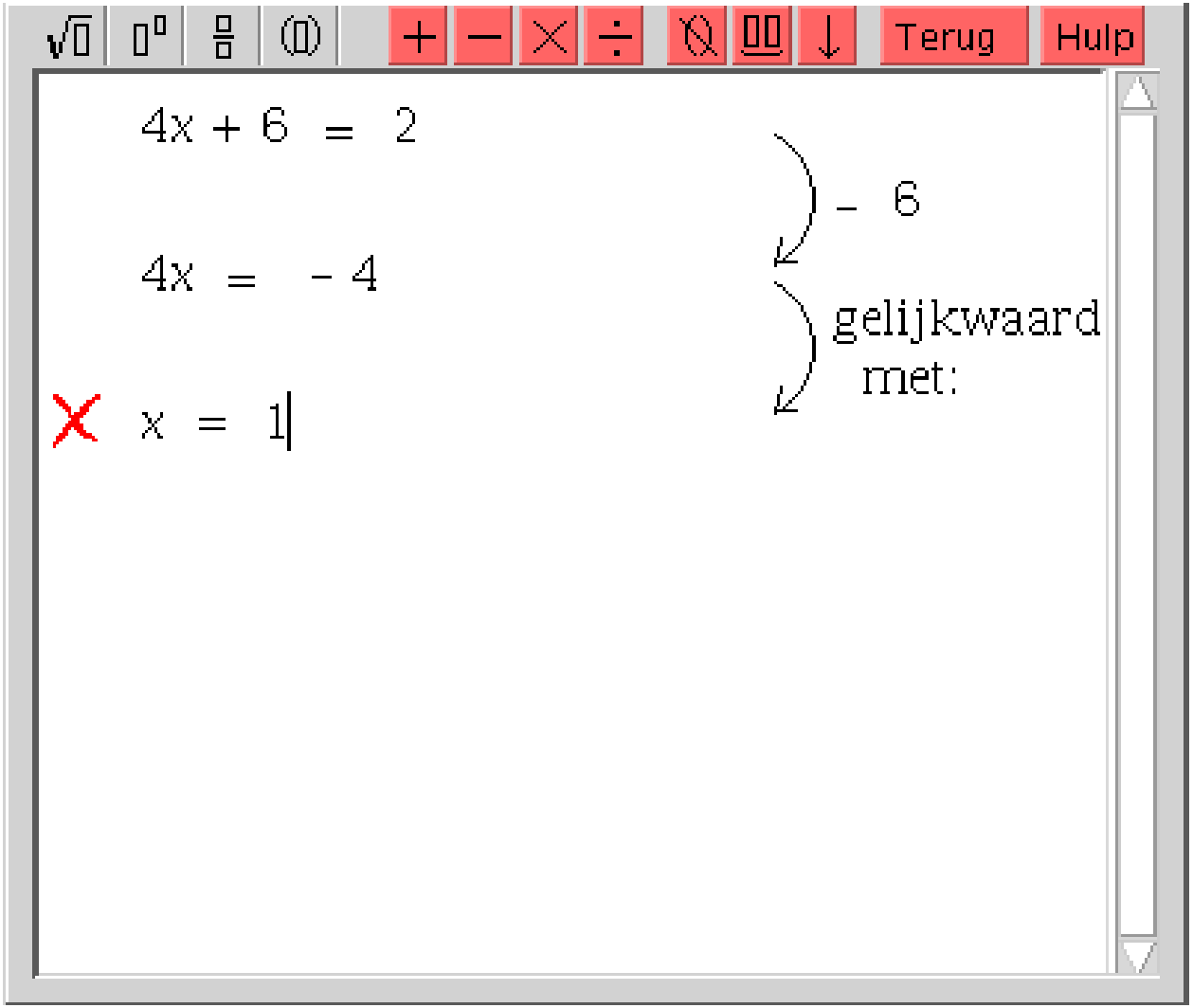
Current solutions



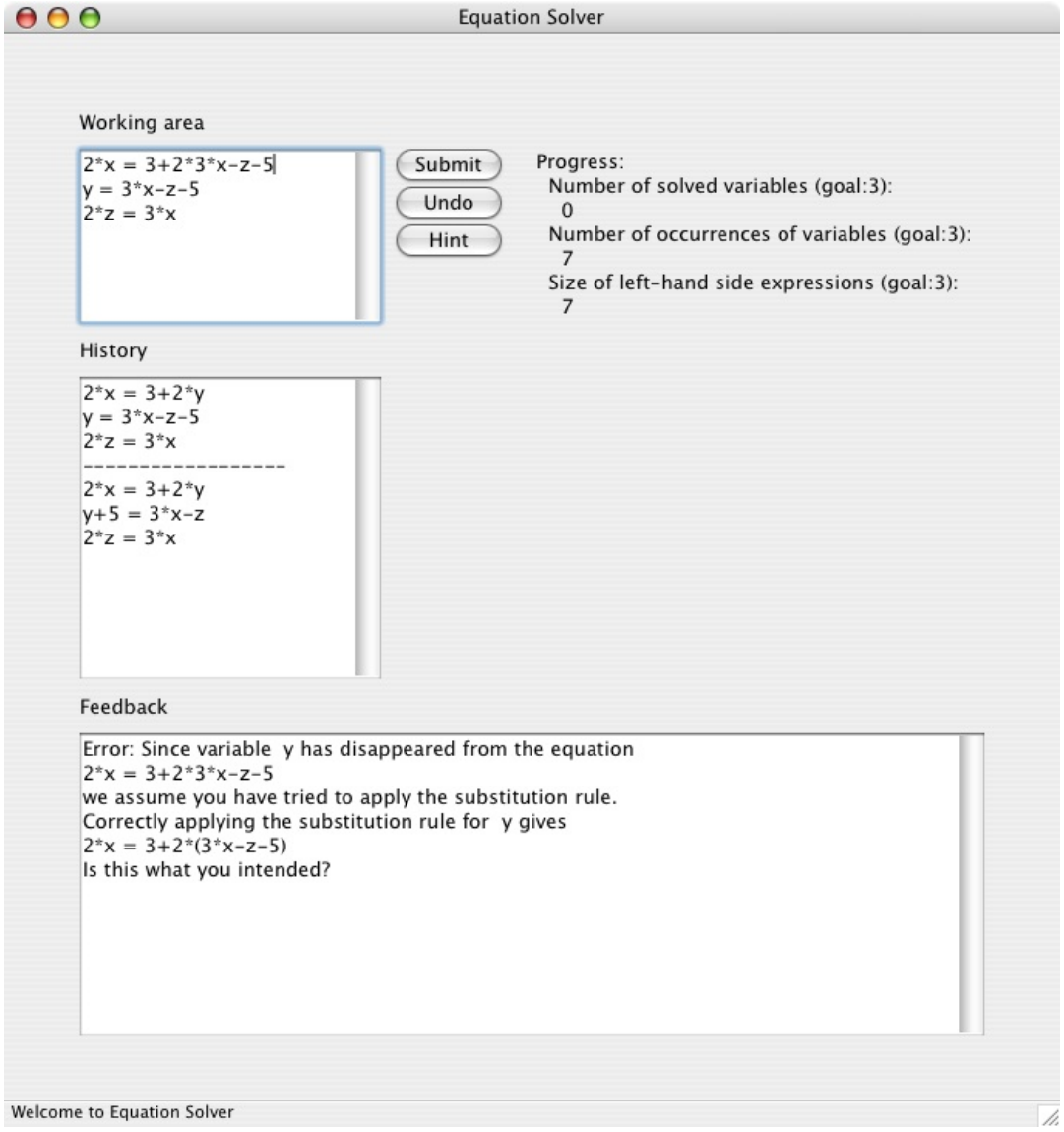
Current solutions



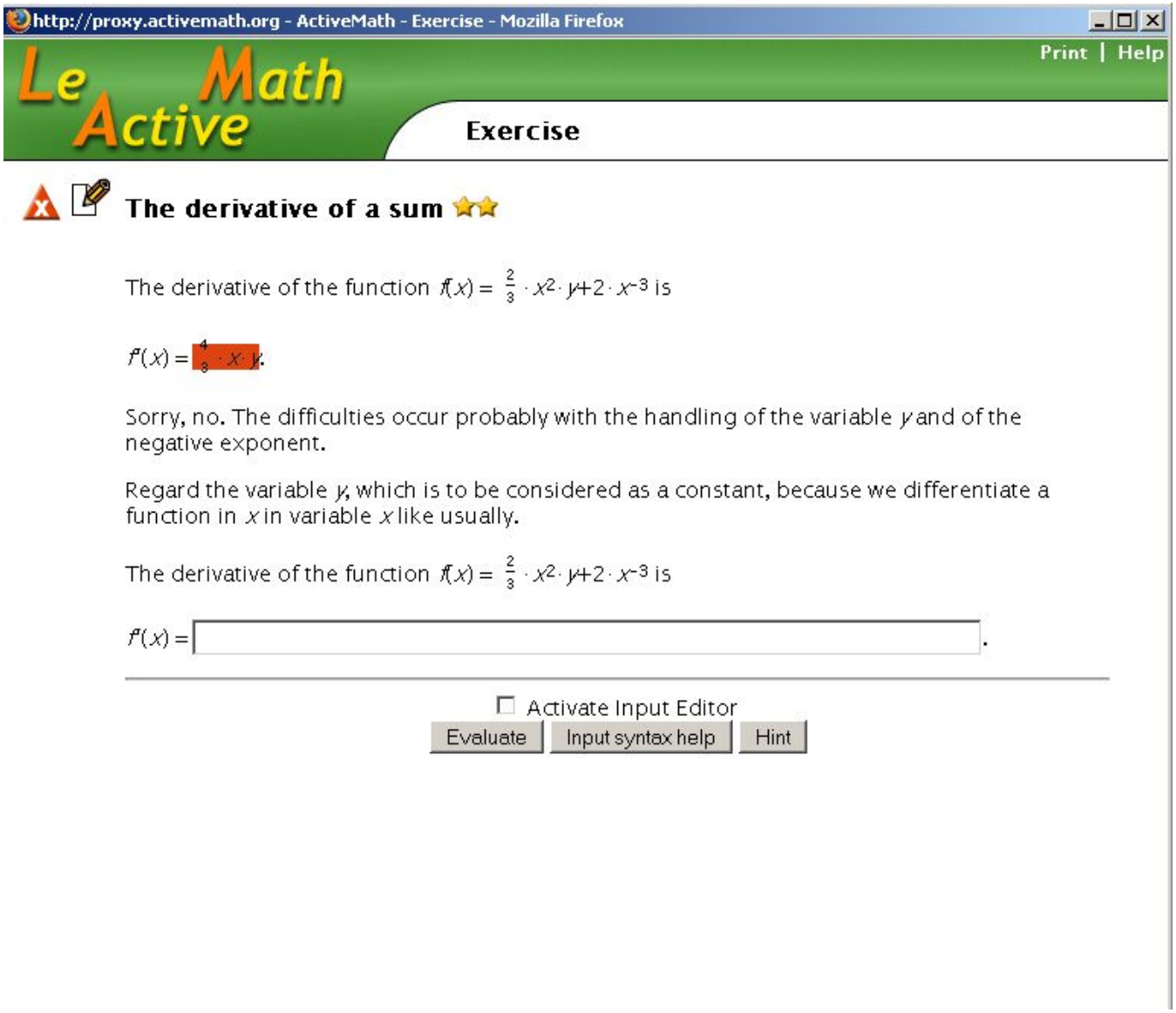
Current solutions



Current solutions



Current solutions



Current solutions

Synopsis

- Feedback is important in learning
- Personal feedback is not feasible without (computerized) help
- Current solutions are either:
  - Only Correct/Incorrect solutions
  - Result of extensive research

Goal

Research question

How can we make the generation of high-quality, domain-specific feedback easier?

Goal

Research question

How can we make the generation of high-quality, domain-specific feedback easier?

Basic idea

Create a generic framework which separates the knowledge of rule-feedback generation from knowledge about the domain.

Requirements

- Should always be able to produce basic feedback
-
-
-

Requirements

- Should always be able to produce basic feedback
- More detailed/complete input leads to better feedback
-
-

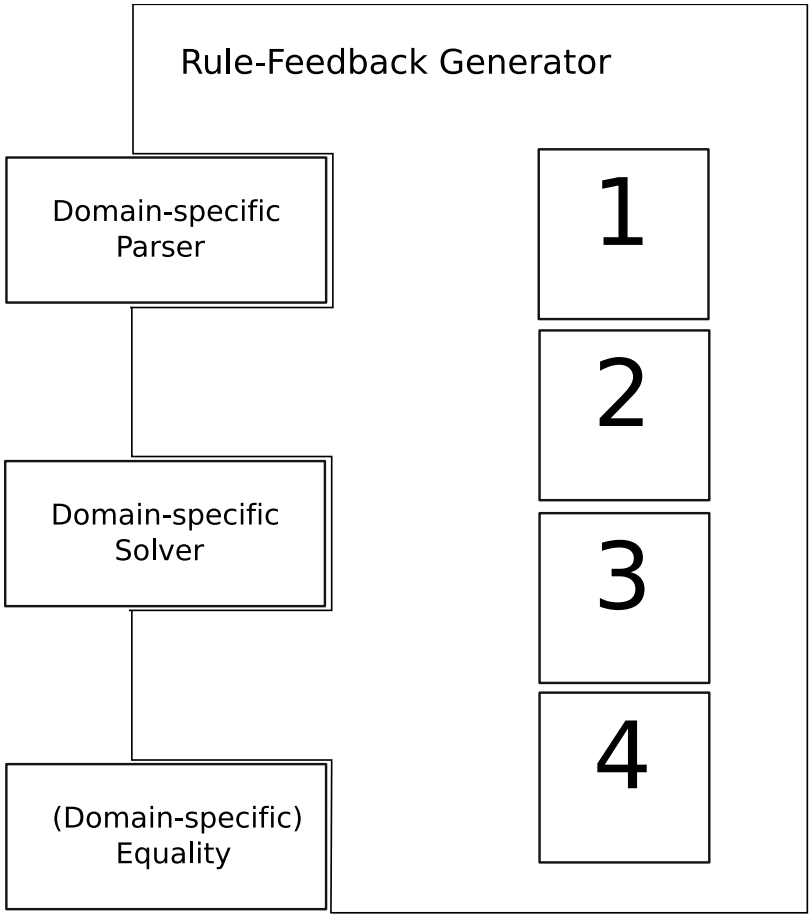
Requirements

- Should always be able to produce basic feedback
- More detailed/complete input leads to better feedback
- Can be instantiated on different domains (with little effort)
-

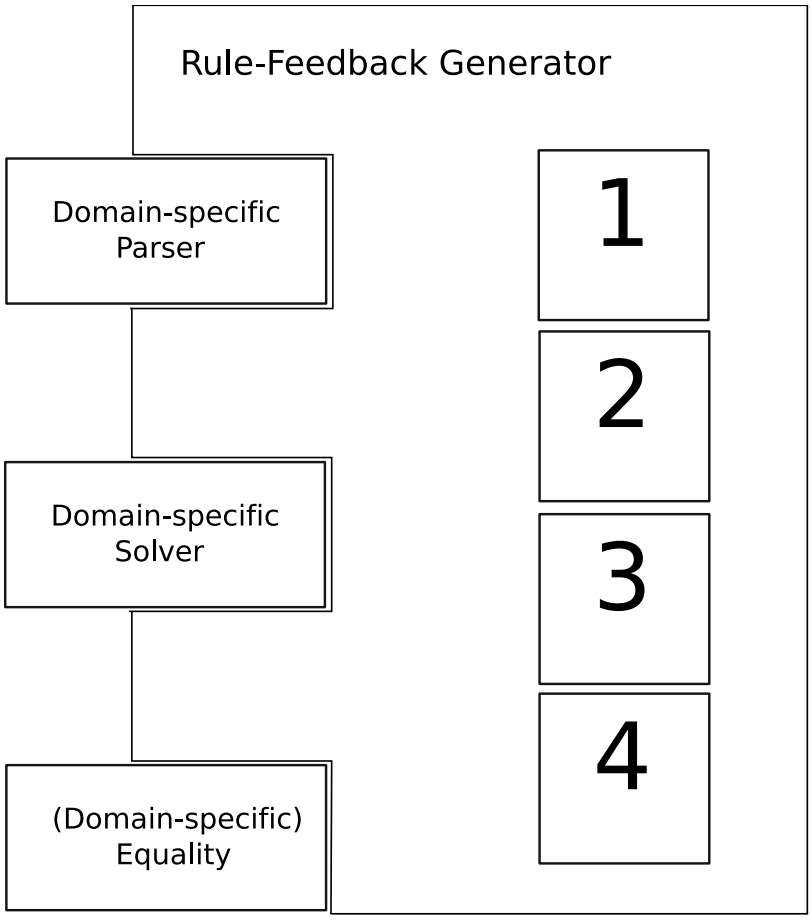
Requirements

- Should always be able to produce basic feedback
- More detailed/complete input leads to better feedback
- Can be instantiated on different domains (with little effort)
- Adaptable to a single class-room (or student!)

Overall design



Overall design

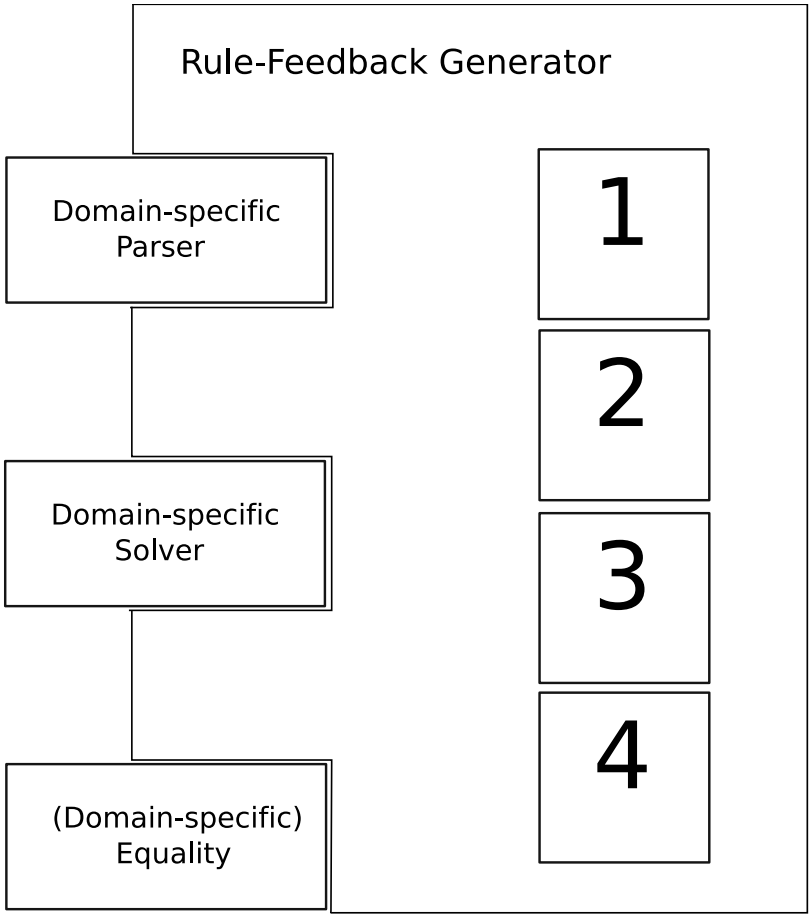


Phase 1:

Input:
CT + PT

Output:
Correct/Incorrect message

Overall design

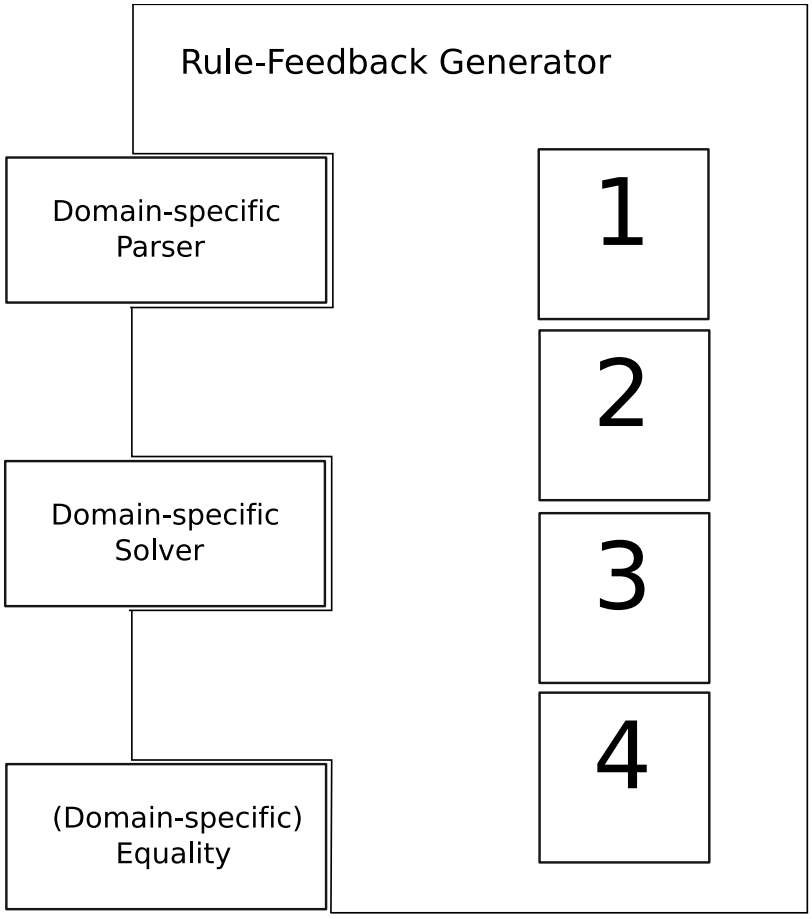


Phase 2:

Input:
CT + PT
+ Student rule

Output:
Correct/Incorrect message-tuple

Overall design

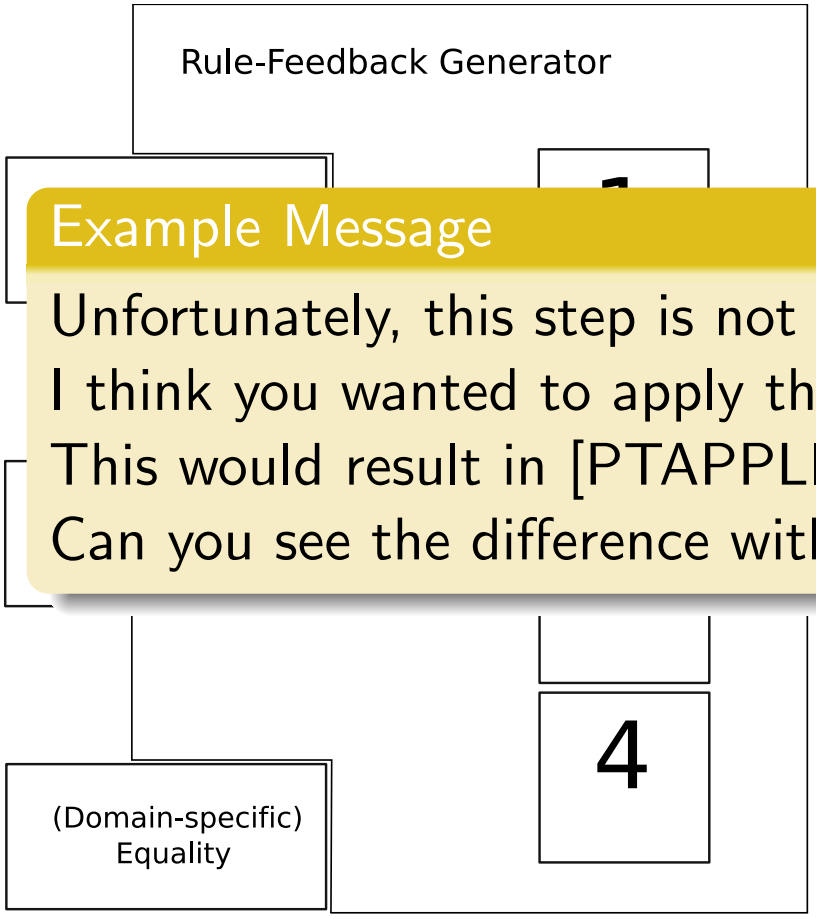


Phase 3:

Input:
CT + PT
+ Allowed rules

Output:
Configurable message, phase level

Overall design



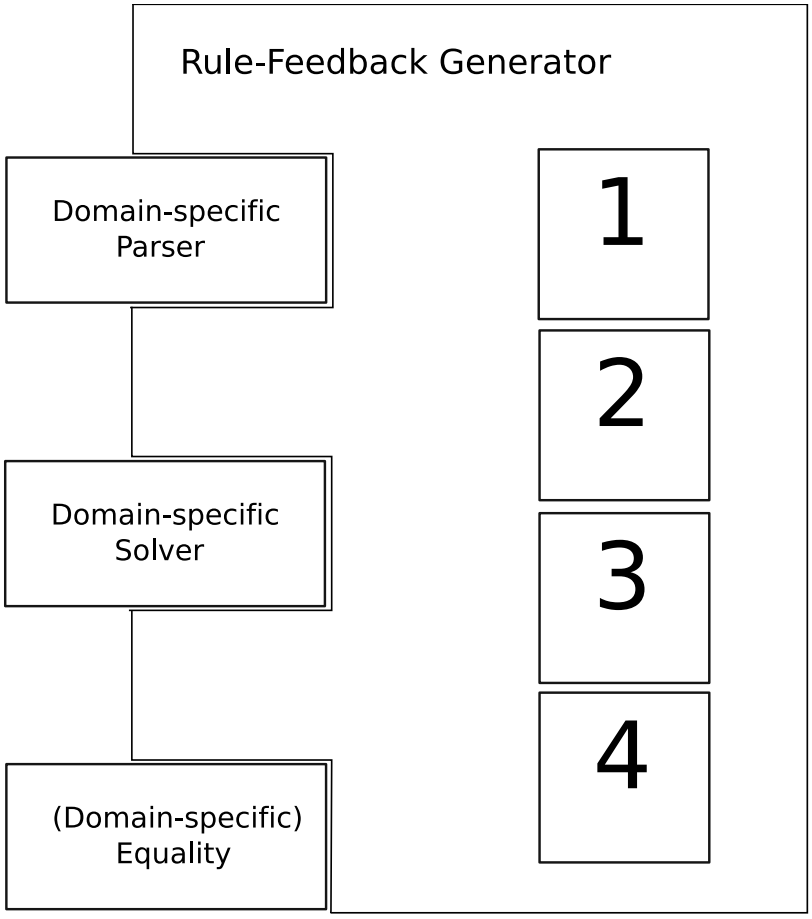
Phase 3:

Example Message

Unfortunately, this step is not correct.
I think you wanted to apply this rule: [RULE].
This would result in [PTAPPLIEDRULE].
Can you see the difference with [CT]?

Output:
Configurable message, phase level

Overall design



Phase 4:

Input:
CT + PT
+ Allowed rules + Buggy rules

Output:
Configurable message, rule level

First phase

Input:	PT, CT
Output:	Correct or Incorrect
Algorithm:	Solve both terms and check whether their results are semantically equal.

First phase

Input:	PT, CT
Output:	Correct or Incorrect
Algorithm:	Solve both terms and check whether their results are semantically equal.

Implementation

```
firstPhase :: RFG a => RFGSolver a -> RFGEqual a
                    -> a -> a -> String
firstPhase solve equal pt ct =
             let resultCt = solve ct
                 resultPt = solve pt
             in if equal resultPt resultCt
                then getConfig Correct
                else getConfig Incorrect
```

First phase

Input:	PT, CT
Output:	Correct or Incorrect
Algorithm:	Solve both terms and check whether their results are semantically equal.

Implementation

```
firstPhase :: RFG a => RFGSolver a -> RFGEqual a
                    -> a -> a -> String
firstPhase solve equal pt ct =
             let resultCt = solve ct
                 resultPt = solve pt
             in if equal resultPt resultCt
                then getConfig Correct
                else getConfig Incorrect
```

First phase

Input:	PT, CT
Output:	Correct or Incorrect
Algorithm:	Solve both terms and check whether their results are semantically equal.

Implementation

```
firstPhase :: RFG a => RFGSolver a -> RFGEqual a
                    -> a -> a -> String
firstPhase solve equal pt ct =
             let resultCt = solve ct
                 resultPt = solve pt
             in if equal resultPt resultCt
                then getConfig Correct
                else getConfig Incorrect
```

Third phase

Input: PT, CT and allowed rules
Output: Configurable message

Third phase

Input: PT, CT and allowed rules
Output: Configurable message

Algorithm

Given two terms, calculate the rewrite rule δ between these terms.
Determine which allowed rule is closest to δ.
Apply rules from the set to the PT if possible and recurse.
The result is the rule which is a closest match.

Third phase

Input: PT, CT and allowed rules
Output: Configurable message

Algorithm

Given two terms, calculate the rewrite rule δ between these terms.
Determine which allowed rule is closest to δ.
Apply rules from the set to the PT if possible and recurse.
The result is the rule which is a closest match.

Third phase

Input: PT, CT and allowed rules
Output: Configurable message

Algorithm

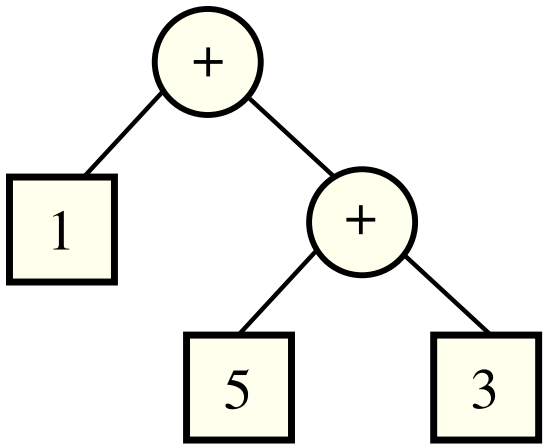
Given two terms, calculate the rewrite rule δ between these terms.
Determine which allowed rule is closest to δ.
Apply rules from the set to the PT if possible and recurse.
The result is the rule which is a closest match.

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

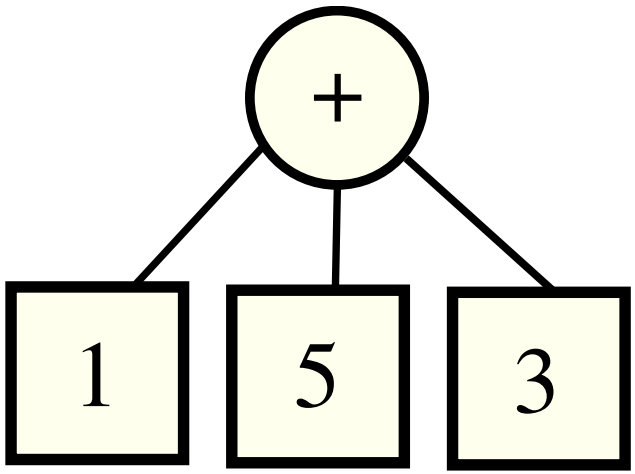
Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$



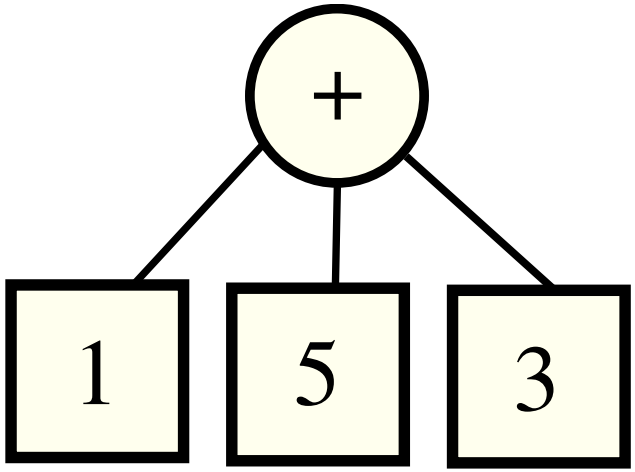
Calculating rewrite rules

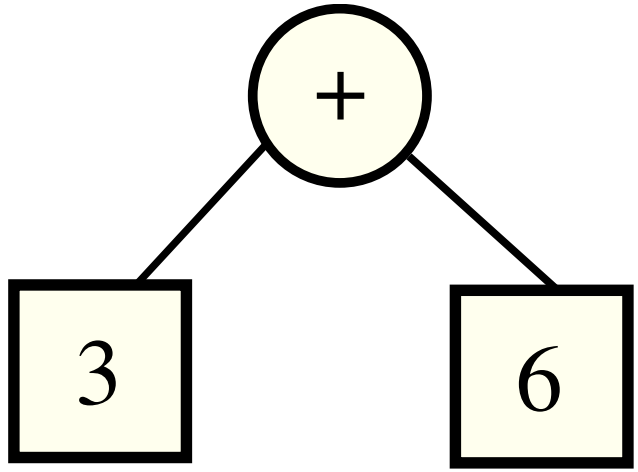
$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$



Calculating rewrite rules

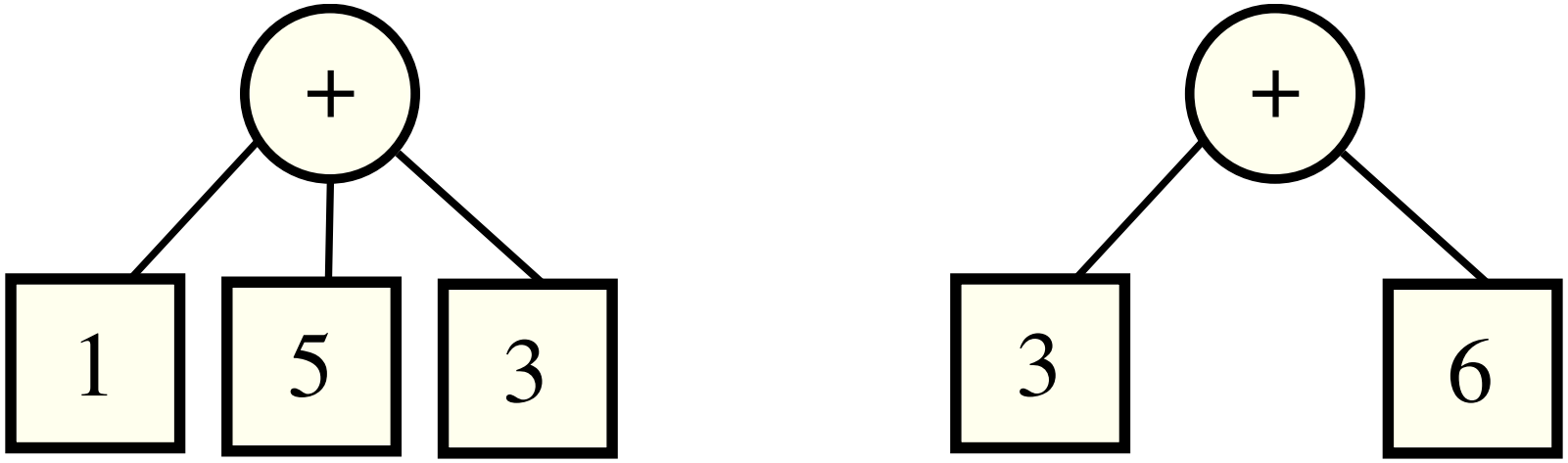
$PT = 1 + 5 + 3$
$CT = 3 + 6$





Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

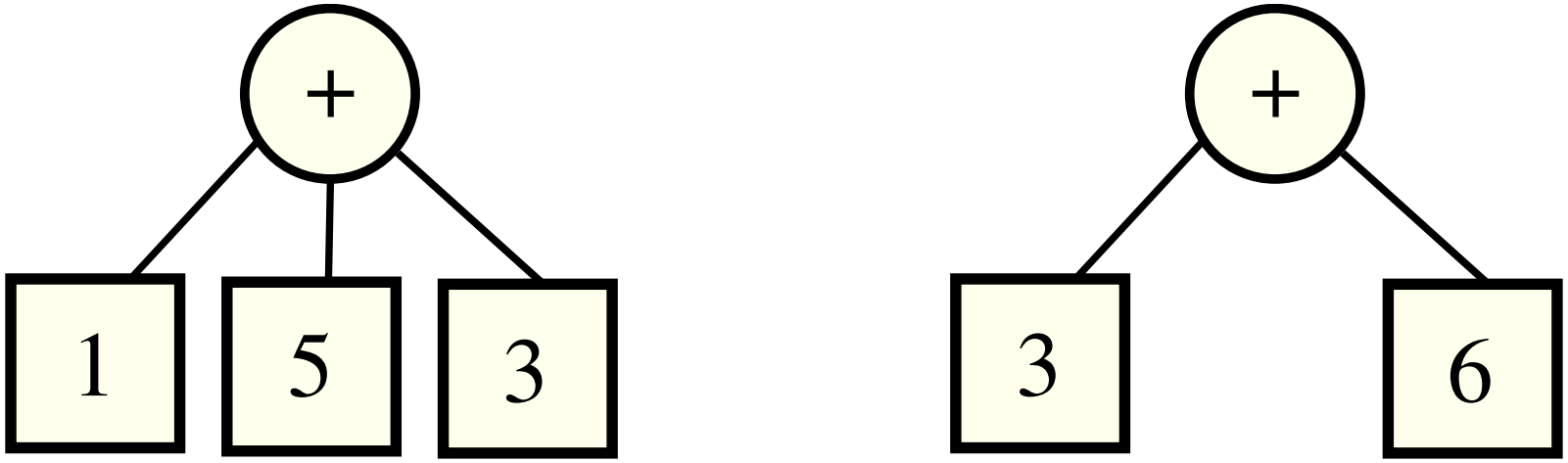


Non-Equal

([1,5,3],[3,6,_])

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$



Equal, Associative

([1,5,3],[3,6,_]) , ([1,5,3],[_,3,6])

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

1

Equal, Associative and Commutative

([1,5,3],[3,6,_]), ([1,5,3],[_,3,6]), ([1,5,3],[6,3,_]),
([1,5,3],[_,6,3]), ([1,3,5],[3,6,_]), ([1,3,5],[_,3,6]),
([1,3,5],[6,3,_]), ([1,3,5],[_,6,3]), ([3,1,5],[3,6,_]),
([3,1,5],[_,3,6]), ([3,1,5],[6,3,_]), ([3,1,5],[_,6,3]),
([3,5,1],[3,6,_]), ([3,5,1],[_,3,6]), ([3,5,1],[6,3,_]),
([3,5,1],[_,6,3]), ([5,3,1],[3,6,_]), ([5,3,1],[_,3,6]),
([5,3,1],[6,3,_]), ([5,3,1],[_,6,3]), ([5,1,3],[3,6,_]),
([5,1,3],[_,3,6]), ([5,1,3],[6,3,_]), ([5,1,3],[_,6,3])

6

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

1

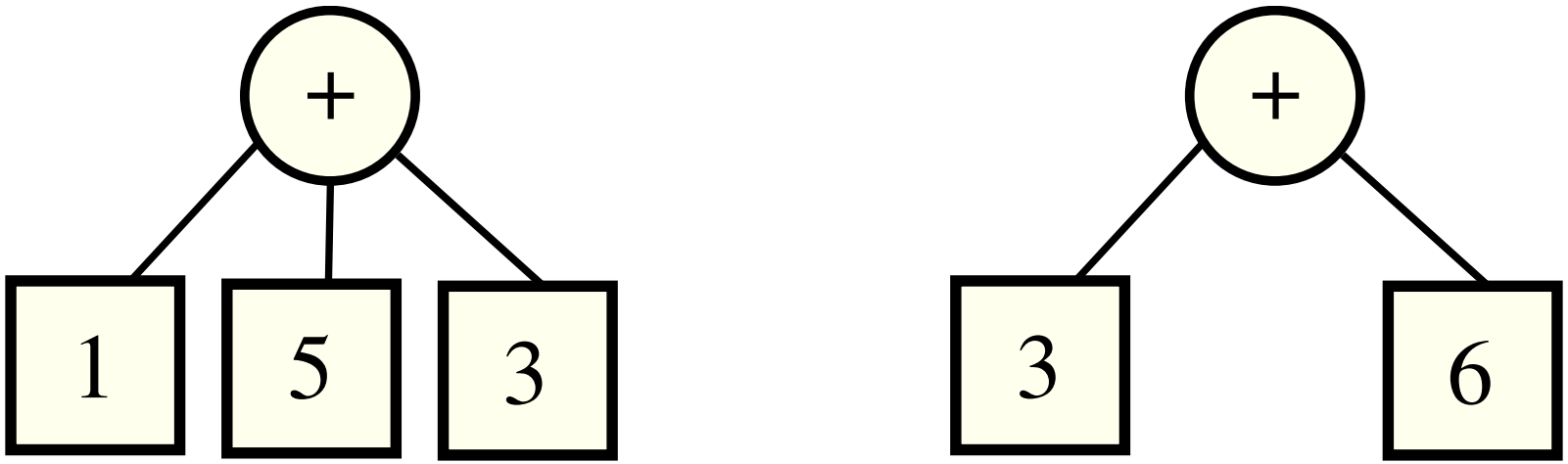
Equal, Associative and Commutative

([1,5,3],[3,6,_]), ([1,5,3],[_,3,6]), ([1,5,3],[6,3,_]),
([1,5,3],[_,6,3]), ([1,3,5],[3,6,_]), ([1,3,5],[_,3,6]),
([1,3,5],[6,3,_]), ([1,3,5],[_,6,3]), ([3,1,5],[3,6,_]),
([3,1,5],[_,3,6]), ([3,1,5],[6,3,_]), ([3,1,5],[_,6,3]),
([3,5,1],[3,6,_]), ([3,5,1],[_,3,6]), ([3,5,1],[6,3,_]),
([3,5,1],[_,6,3]), ([5,3,1],[3,6,_]), ([5,3,1],[_,3,6]),
([5,3,1],[6,3,_]), ([5,3,1],[_,6,3]), ([5,1,3],[3,6,_]),
([5,1,3],[_,3,6]), ([5,1,3],[6,3,_]), ([5,1,3],[_,6,3])

6

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

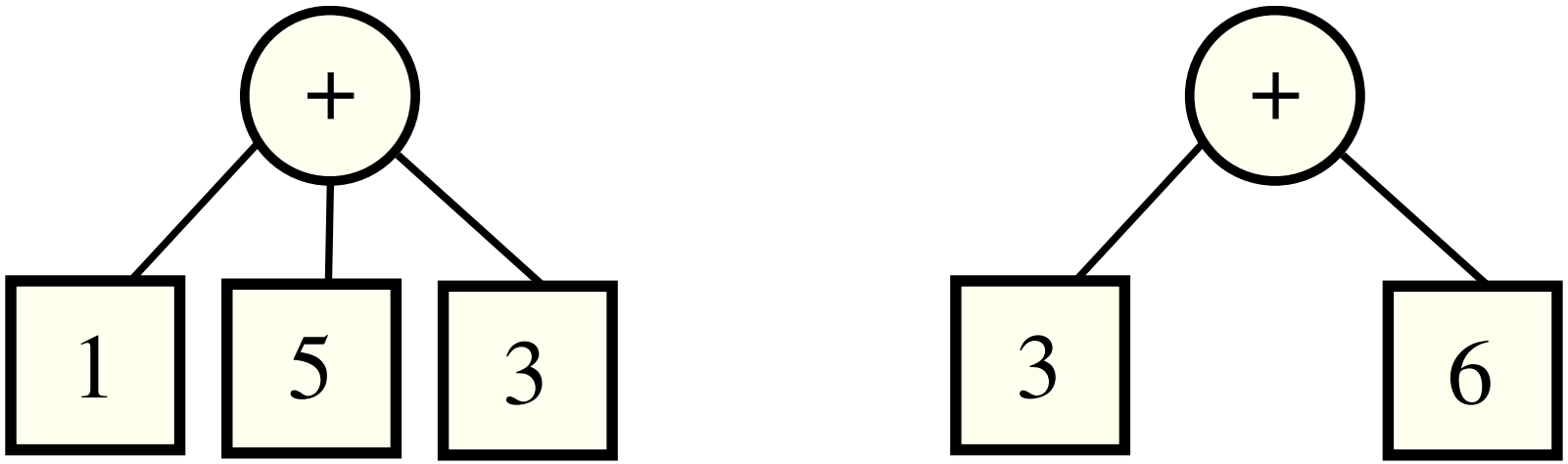


Result:

([1,5],[6])

Calculating rewrite rules

$$PT = 1 + 5 + 3$$
$$CT = 3 + 6$$

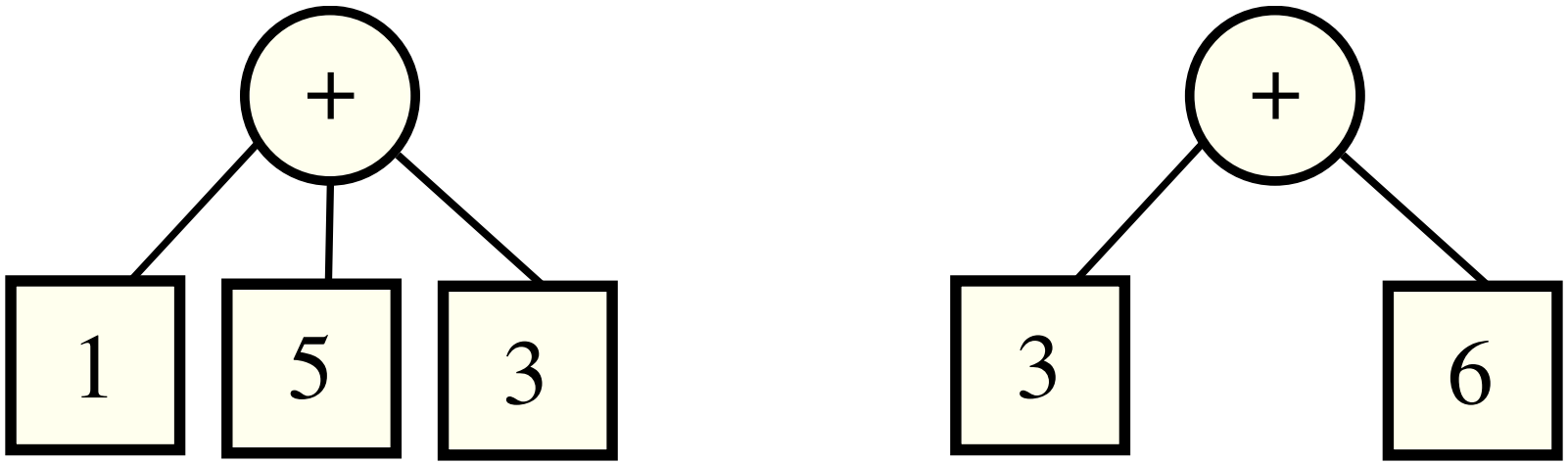


Result:

$\delta = 1 + 5 \Rightarrow +6$

Calculating rewrite rules

$PT = 1 + 5 + 3$
$CT = 3 + 6$



Result:

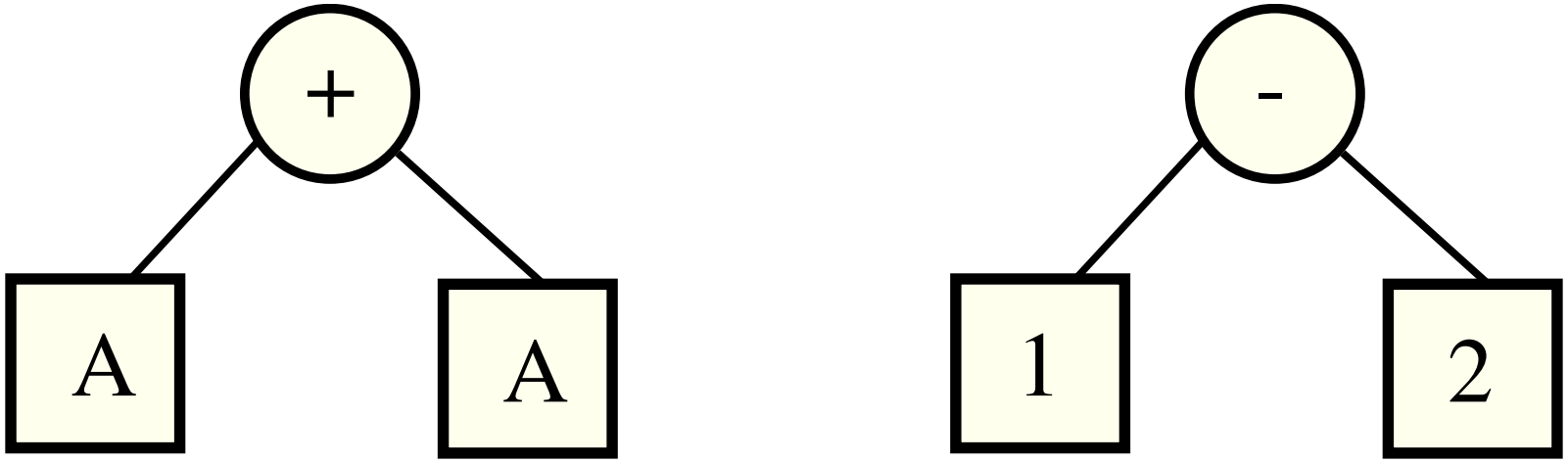
$\delta = 1 + 5 \Rightarrow 6$

Defining the distance

Distance between rules:

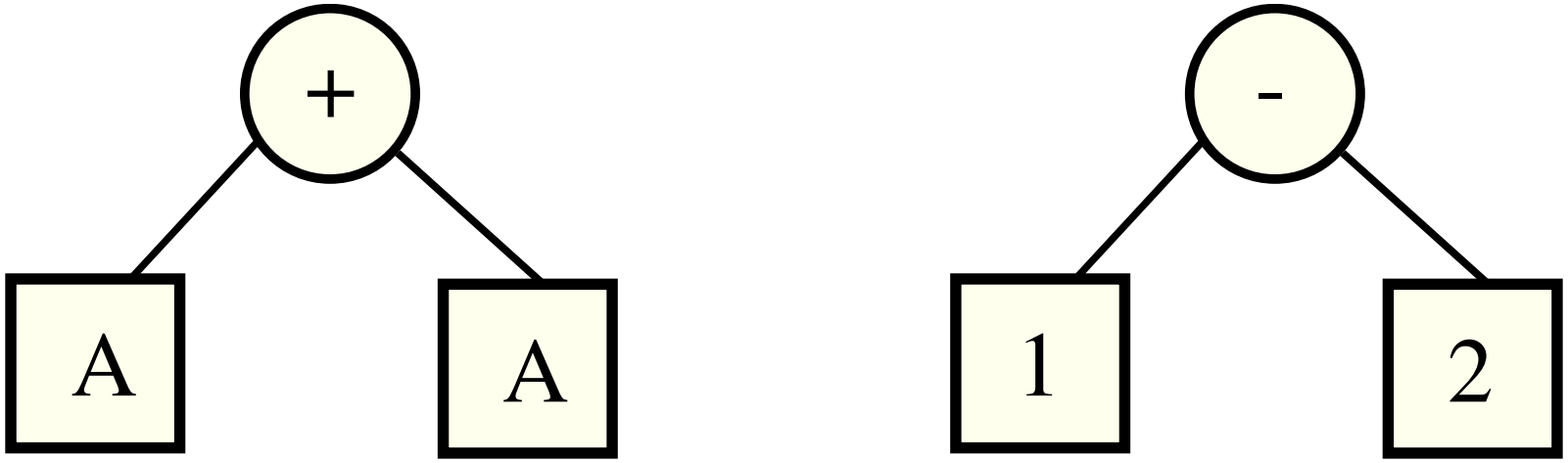
Defining the distance

Distance between terms:



Defining the distance

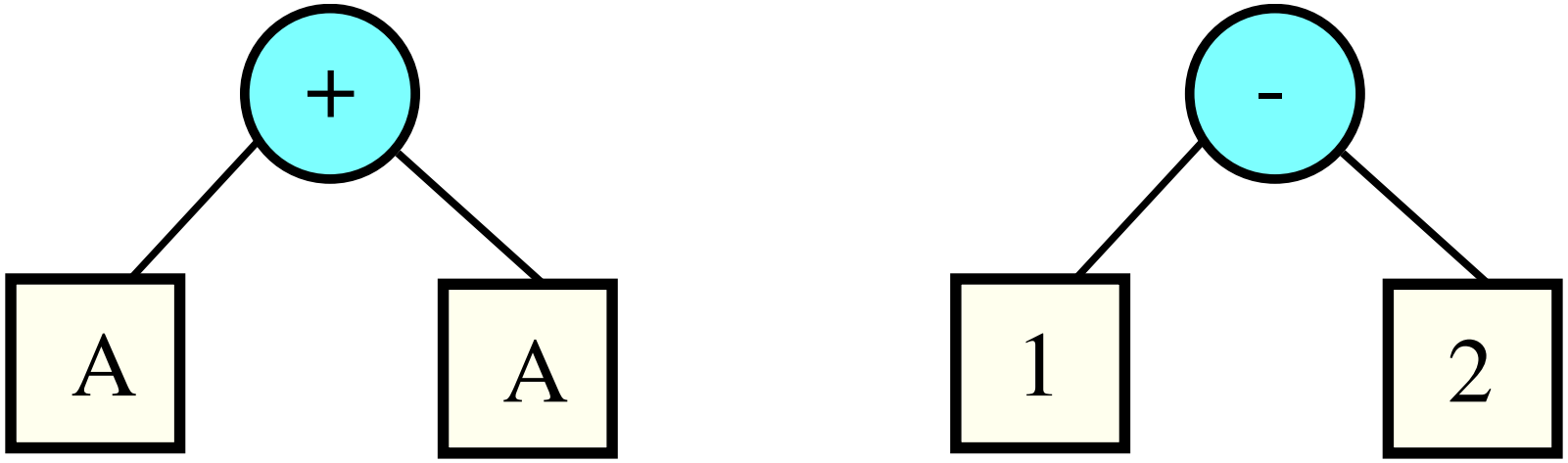
Distance between terms:



Environment $= \{\}$
Distance $= 0$

Defining the distance

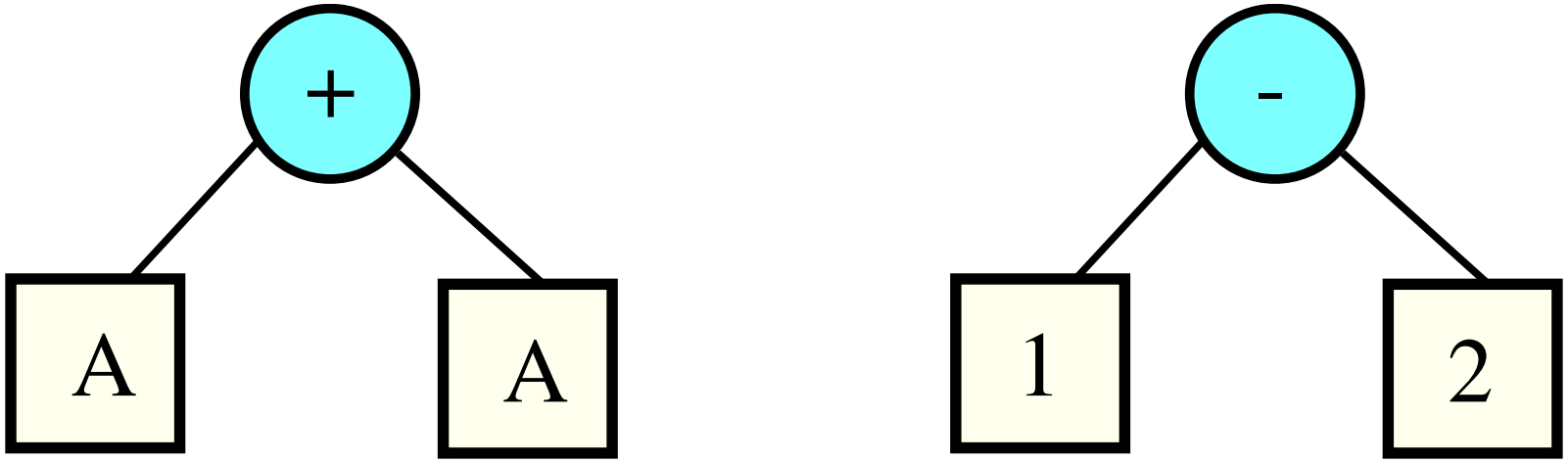
Distance between terms:



Environment $= \{\}$
Distance $= 0$

Defining the distance

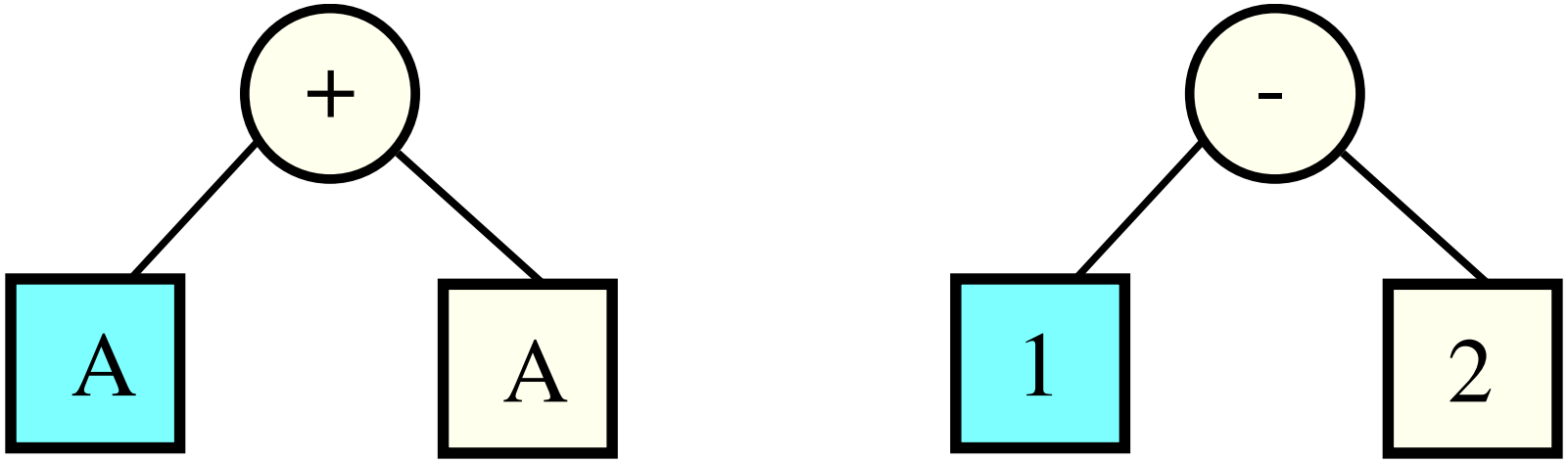
Distance between terms:



Environment $= \{\}$
Distance $= 2$

Defining the distance

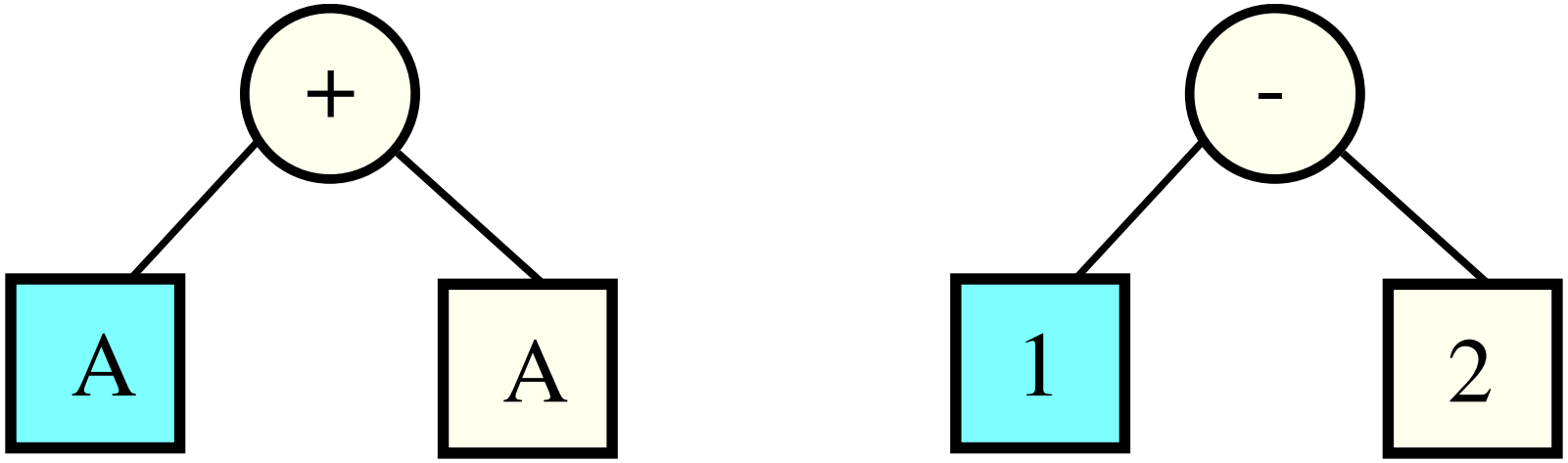
Distance between terms:



Environment $= \{\}$
Distance $= 2$

Defining the distance

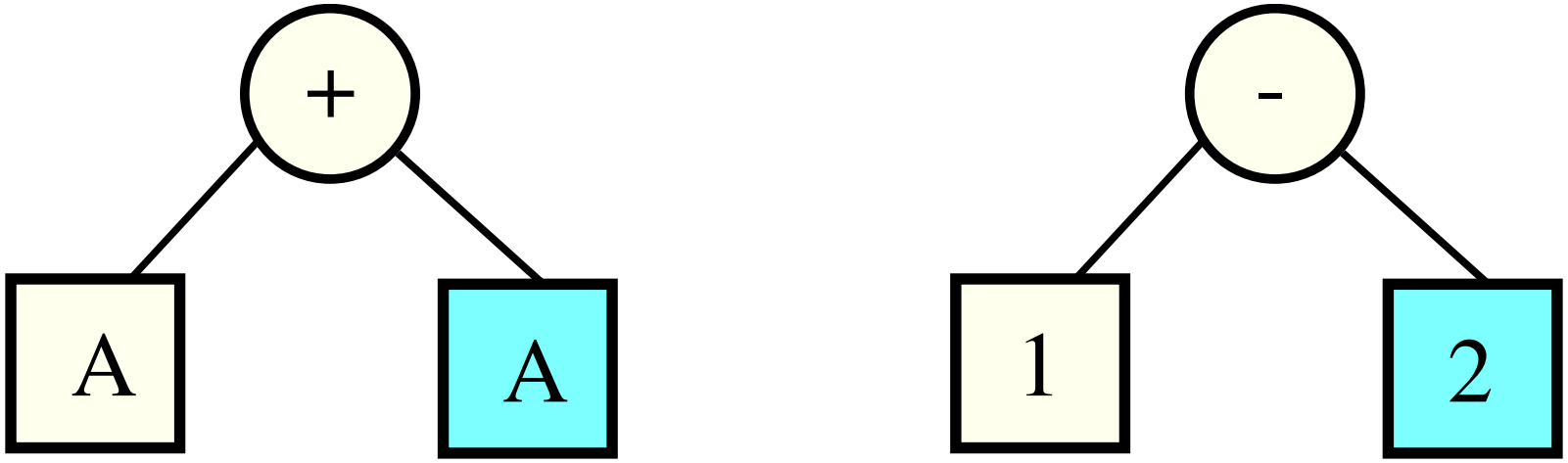
Distance between terms:



Environment $= \{(A, 1)\}$
Distance $= 2$

Defining the distance

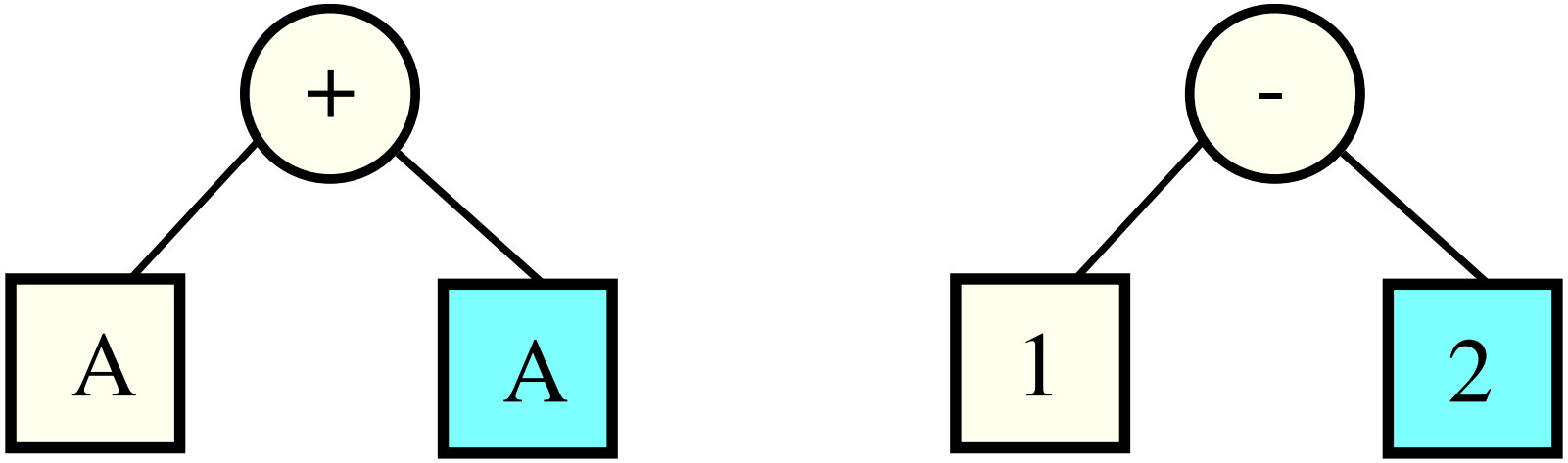
Distance between terms:



Environment $= \{(A, 1)\}$
Distance $= 2$

Defining the distance

Distance between terms:



Environment $= \{(A, 1)\}$
Distance $= 4$

Example

$$PT = (a \vee b) \rightarrow c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned}
&\text{Rules} = \{ \\
&\quad \text{ImpElimination} : \\
&\qquad A \rightarrow B \Rightarrow \neg(A) \vee B \\
&, \ \text{MorganOr} : \\
&\qquad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\
&\}
\end{aligned}$$

Example

$$PT = (a \vee b) \rightarrow c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned} &\text{Rules} = \{ \\ &\quad \text{ImpElimination} : \\ &\qquad A \rightarrow B \Rightarrow \neg(A) \vee B \\ &, \text{MorganOr} : \\ &\qquad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ &\} \end{aligned}$$

Calculated Rule

$$\delta = (a \vee b) \rightarrow c \Rightarrow (\neg a \wedge b) \vee c$$

Example

$$PT = (a \vee b) \rightarrow c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned} Rules = \{ & \\ & ImpElimination : \\ & \quad A \rightarrow B \Rightarrow \neg(A) \vee B \\ , & \ MorganOr : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \} & \end{aligned}$$

Calculated Rule

$\delta = (a \vee b) \rightarrow c \Rightarrow (\neg a \wedge b) \vee c$

Distances

$dist(\delta, ImpElimination) = 9$
$dist(\delta, MorganOr) = 15$

Example

$$PT = (a \vee b) \rightarrow c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned} &\text{Rules} = \{ \\ &\quad \text{ImpElimination} : \\ &\qquad A \rightarrow B \Rightarrow \neg(A) \vee B \\ &, \text{ MorganOr} : \\ &\qquad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ &\} \end{aligned}$$

Calculated Rule

$\delta = (a \vee b) \rightarrow c \Rightarrow (\neg a \wedge b) \vee c$

Distances

$dist(\delta, ImpElimination) = 9$
$dist(\delta, MorganOr) = 15$

Example

PT' $= \neg(a \vee b) \vee c$

CT $= (\neg a \wedge b) \vee c$

$$
\begin{aligned}
&\text{Rules} = \{ \\
&\quad \text{ImpElimination} : \\
&\qquad A \rightarrow B \Rightarrow \neg(A) \vee B \\
&, \ \text{MorganOr} : \\
&\qquad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\
&\}
\end{aligned}
$$

Example

$$\text{PT'} = \neg(a \vee b) \vee c$$

$$\text{CT} = (\neg a \wedge b) \vee c$$

$$\begin{aligned}\text{Rules} = \{ & \\ & \text{ImpElimination} : \\ & \quad A \to B \Rightarrow \neg(A) \vee B \\ , & \text{MorganOr} : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \}&\end{aligned}$$

Calculated Rule

$$\delta = \neg(a \vee b) \Rightarrow (\neg a \wedge b)$$

Example

$$\text{PT'} = \neg(a \vee b) \vee c$$

$$\text{CT} = (\neg a \wedge b) \vee c$$

$$\begin{aligned}\text{Rules} = \{ & \\ & \text{ImpElimination} : \\ & \quad A \to B \Rightarrow \neg(A) \vee B \\ , & \text{MorganOr} : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \}&\end{aligned}$$

Calculated Rule

$\delta = \neg(a \vee b) \Rightarrow (\neg a \wedge b)$

Distances

$dist(\delta, ImpElimination) = 9$
$dist(\delta, MorganOr) = 4$

Example

$$\text{PT'} = \neg(a \vee b) \vee c$$

$$\text{CT} = (\neg a \wedge b) \vee c$$

$$\begin{aligned}\text{Rules} = \{ & \\ & \text{ImpElimination} : \\ & \quad A \to B \Rightarrow \neg(A) \vee B \\ & , \text{MorganOr} : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \}&\end{aligned}$$

Calculated Rule

$\delta = \neg(a \vee b) \Rightarrow (\neg a \wedge b)$

Distances

$dist(\delta, ImpElimination) = 9$
$dist(\delta, MorganOr) = 4$

Example

$$PT'' = (\neg a \wedge \neg b) \vee c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned}
&\text{Rules} = \{ \\
&\quad \text{ImpElimination} : \\
&\qquad A \to B \Rightarrow \neg(A) \vee B \\
&, \text{ MorganOr} : \\
&\qquad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\
&\}
\end{aligned}$$

Example

$$PT'' = (\neg a \wedge \neg b) \vee c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned} Rules = \{ & \\ & ImpElimination : \\ & \quad A \rightarrow B \Rightarrow \neg(A) \vee B \\ , & \ MorganOr : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \} & \end{aligned}$$

Calculated Rule

$\delta = \neg b \Rightarrow b$

Example

$$PT'' = (\neg a \wedge \neg b) \vee c$$

$$CT = (\neg a \wedge b) \vee c$$

$$\begin{aligned} Rules = \{ & \\ & ImpElimination : \\ & \quad A \rightarrow B \Rightarrow \neg(A) \vee B \\ , & \ MorganOr : \\ & \quad \neg(A \vee B) \Rightarrow \neg(A) \wedge \neg(B) \\ \} & \end{aligned}$$

Calculated Rule

$\delta = \neg b \Rightarrow b$

Distances

$dist(\delta, ImpElimination) = 9$
$dist(\delta, MorganOr) = 10$

Fourth phase

M. Hennecke
Online Diagnose in intelligenten mathematischen Lehr-Lern-Systemen.
PhD thesis, Hildesheim University, 1999.

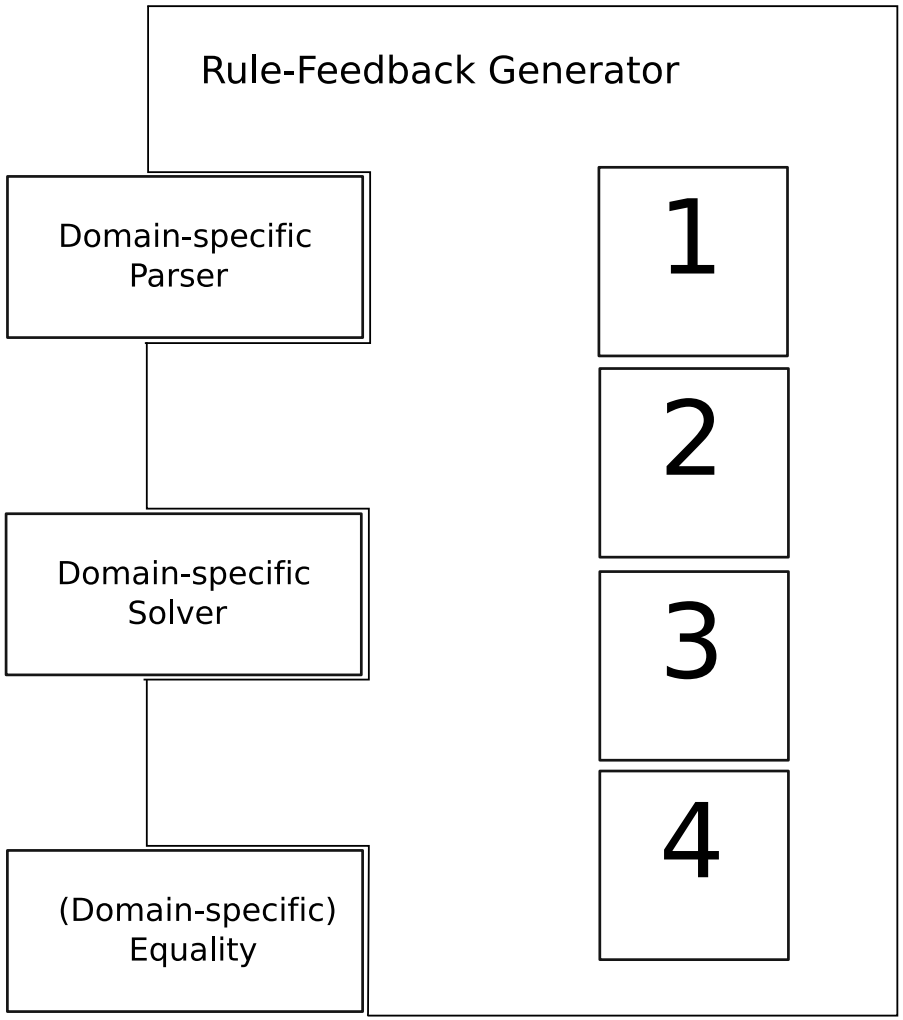
Fourth phase

M. Hennecke
Online Diagnose in intelligenten mathematischen Lehr-Lern-Systemen.
PhD thesis, Hildesheim University, 1999.

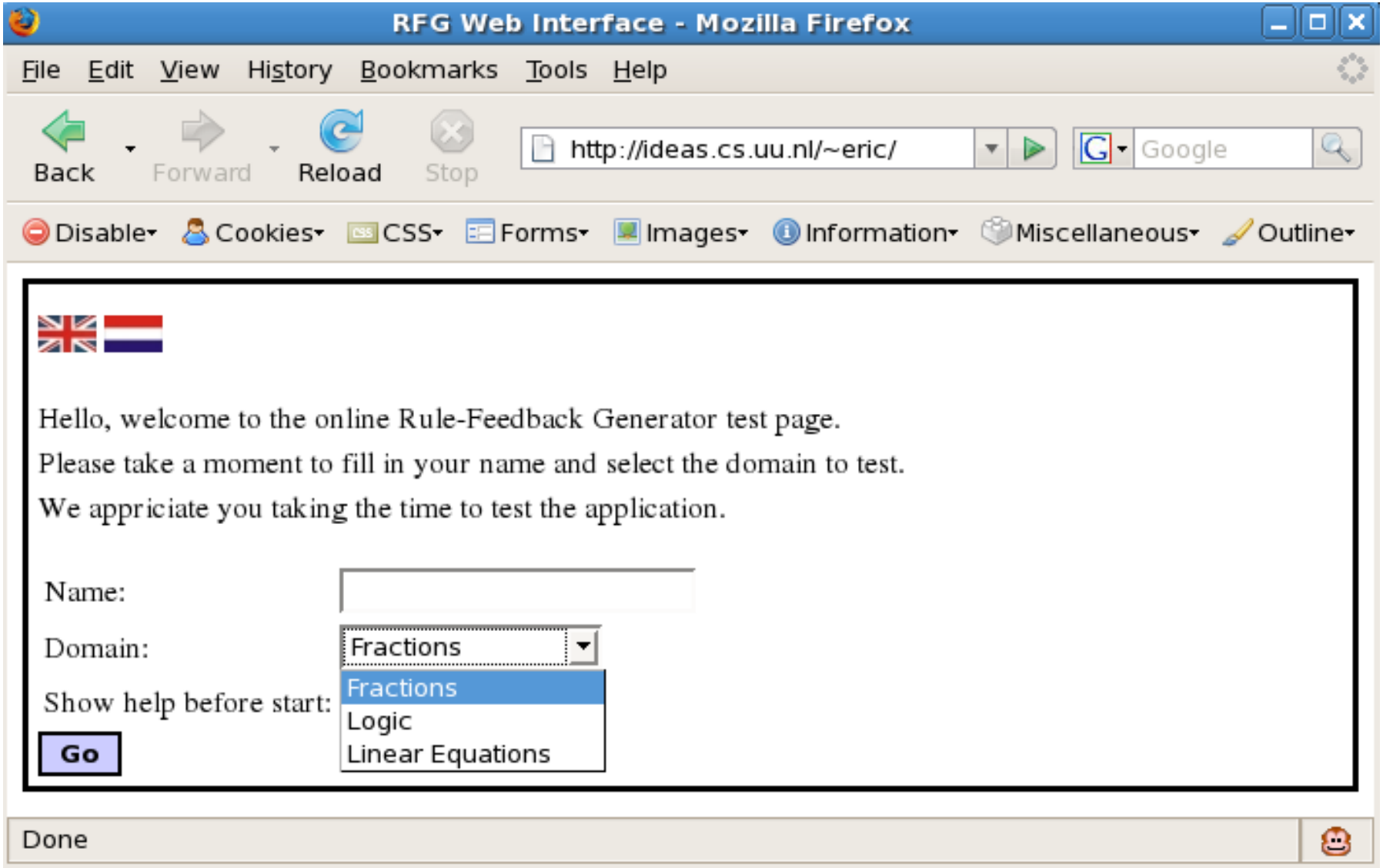
Example

$\frac{A}{B} + \frac{C}{D} \Rightarrow \frac{A+C}{B+D}$
: AddError, You can only add fractions with equal denominators
: 0.3

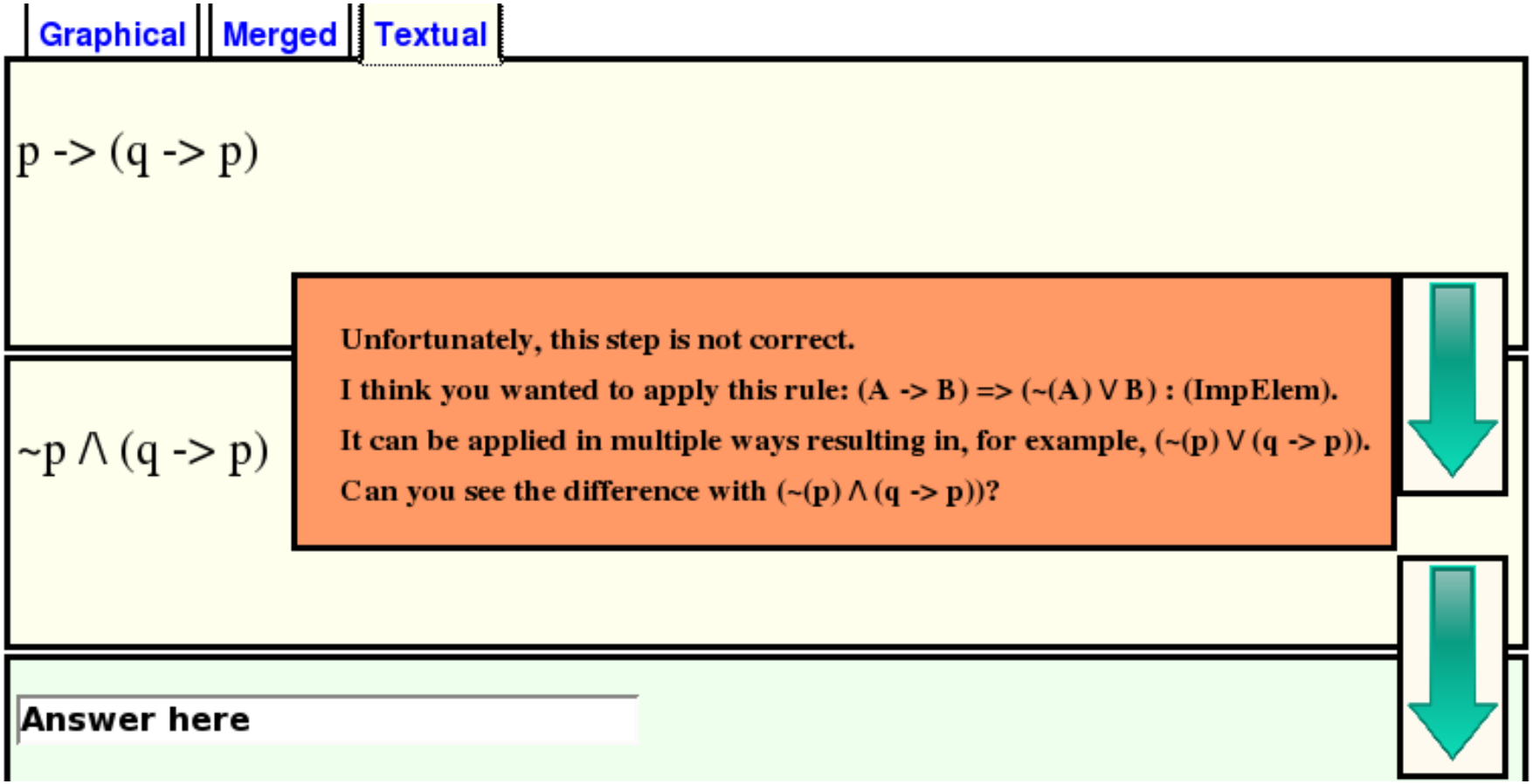
Overview


Rule-Feedback Generator
Domain-specific Parser
Domain-specific Solver
(Domain-specific) Equality
1
2
3
4

Receiving feedback



Feedback message



Wrong Guesses

EqElem	: $A \leftrightarrow B$	$\longrightarrow$	$(A \wedge B) \vee (\neg(A) \wedge \neg(B))$
ImpElem	: $A \rightarrow B$	$\longrightarrow$	$\neg(A) \vee B$
MorganA	: $\neg(A \wedge B)$	$\longrightarrow$	$\neg(A) \vee \neg(B)$
NotNot	: $\neg\neg A$	$\longrightarrow$	A
DistAO	: $A \wedge (B \vee C)$	$\longrightarrow$	$(A \wedge B) \vee (A \wedge C)$
IdemA	: $A \wedge A$	$\longrightarrow$	A

Wrong Guesses

EqElem	: $A \leftrightarrow B$	$\rightarrow$	$(A \wedge B) \vee (\neg(A) \wedge \neg(B))$
ImpElem	: $A \rightarrow B$	$\rightarrow$	$\neg(A) \vee B$
MorganA	: $\neg(A \wedge B)$	$\rightarrow$	$\neg(A) \vee \neg(B)$
NotNot	: $\neg\neg A$	$\rightarrow$	A
DistAO	: $A \wedge (B \vee C)$	$\rightarrow$	$(A \wedge B) \vee (A \wedge C)$
IdemA	: $A \wedge A$	$\rightarrow$	A

PT	CT	Expected rule	Actual output
$a \rightarrow (b \wedge c)$	$a \vee (b \wedge c)$	ImpElem	NotNot
$\neg\neg a \leftrightarrow b$	$(\neg a \wedge b) \vee (\neg\neg\neg a \vee \neg b)$	EqElem	MorganA
$a \vee c \leftrightarrow b$	$(a \vee c \wedge b) \vee \neg(a \vee c) \wedge \neg b$	EqElem	IdemA
$a \leftrightarrow b \wedge (c \vee d)$	$(a \leftrightarrow b \wedge c) \vee (a \leftrightarrow b \wedge d)$	EqElem	DistAO

Definition of Rules

Multiplication	:	$(A/B) * (C/D)$	$\Rightarrow$	$(A * C) / (B * D)$
Division	:	$(A/B) / (C/D)$	$\Rightarrow$	$(A/B) * (D/C)$
SameMixed	:	$A\|B/B$	$\Rightarrow$	$A + 1$
AddFractions	:	$(A/B) + (C/B)$	$\Rightarrow$	$(A + C) / B$
SubFractions	:	$(A/B) - (C/B)$	$\Rightarrow$	$(A - C) / B$

Definition of Rules

Multiplication	:	$(A/B) * (C/D)$	$\Rightarrow$	$(A * C) / (B * D)$
Division	:	$(A/B) / (C/D)$	$\Rightarrow$	$(A/B) * (D/C)$
SameMixed	:	$A\|B/B$	$\Rightarrow$	$A + 1$
AddFractions	:	$(A/B) + (C/B)$	$\Rightarrow$	$(A + C) / B$
SubFractions	:	$(A/B) - (C/B)$	$\Rightarrow$	$(A - C) / B$

SolveMin	:	$A - B$	$\Rightarrow$	C *where* C	$:=$	$solve(A - B)$
SolveAdd	:	$A + B$	$\Rightarrow$	C *where* C	$:=$	$solve(A + B)$
SolveMul	:	$A * B$	$\Rightarrow$	C *where* C	$:=$	$solve(A * B)$

Conclusion

- Partial/Configurable approach is possible
- Adding an extra domain takes little effort
- Combining techniques leads to surprisingly good results
- Already useful, both for research as well as practical

Resources

Thesis page:
`http://www.cs.uu.nl/wiki/Students/EricBouwersThesisPage`

Prototype:
`http://ideas.cs.uu.nl/~eric/`

Contact me:
`EricBouwers@gmail.com`