Mechanised Justification in "Systems that Explain Themselves" for Mathematics Education

Walther Neuper

IICM, Institute for Computer Media,
University of Technology.
Graz, Austria

eduTPS: Working Group on Justification in Doing Math
at CADGME, Coimbra, Portugal
June TODO, 2018

Outline

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle
- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle
- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle

- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle

- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle

- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, *ISAC*
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/*ISAC* will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle
- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle
- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

"Systems that Explain Themselves"?

- Systems for proofs: well-known theorem provers (TP), e.g. Coq, PVS, HOL, Isabelle have
  - math knowledge deduced from first principles (axioms)
  - so, elements of math **knowledge** "explain themselves"
  - **usage** for proof does **not** explain itself

  $\longrightarrow$ short demo of Isabelle
- Systems for engineering mathematics: only prototypes, e.g. 4ferries (by R.J. Back), Mathtoys, $\mathcal{ISAC}$
  - need to build upon TPs (justifications !)
  - need to step-wise construct problem solutions
  - need to support modularisation into sub-problems
  - . . . such that **usage** and **knowledge** "explain themselves"

  $\longrightarrow$ Isabelle/$\mathcal{ISAC}$ will serve as example

Outline

Term Language

The demo has shown . . .

- principal benefits
  - uniformity over all domains of mathematics
  - type system efficiently excludes ambiguities
  - clear description of functions and respective rules
- added value of implementation
  - a formula's elements are connected with definitions
  - types are transparent by mouse pointer
  - feedback to input of formulas
  - structure of formulas, i.e. sub-terms are transparent
  - internal representation adaptable to engineers' needs

Term Language

The demo has shown . . .

- principal benefits
  - uniformity over all domains of mathematics
  - type system efficiently excludes ambiguities
  - clear description of functions and respective rules
- added value of implementation
  - a formula's elements are connected with definitions
  - types are transparent by mouse pointer
  - feedback to input of formulas
  - structure of formulas, i.e. sub-terms are transparent
  - internal representation adaptable to engineers' needs

Term Language

The demo has shown . . .

- principal benefits
  - uniformity over all domains of mathematics
  - type system efficiently excludes ambiguities
  - clear description of functions and respective rules
- added value of implementation
  - a formula's elements are connected with definitions
  - types are transparent by mouse pointer
  - feedback to input of formulas
  - structure of formulas, i.e. sub-terms are transparent
  - internal representation adaptable to engineers' needs

Outline

Proof Language . . .

. . . adapts to conventions of engineering mathematics:

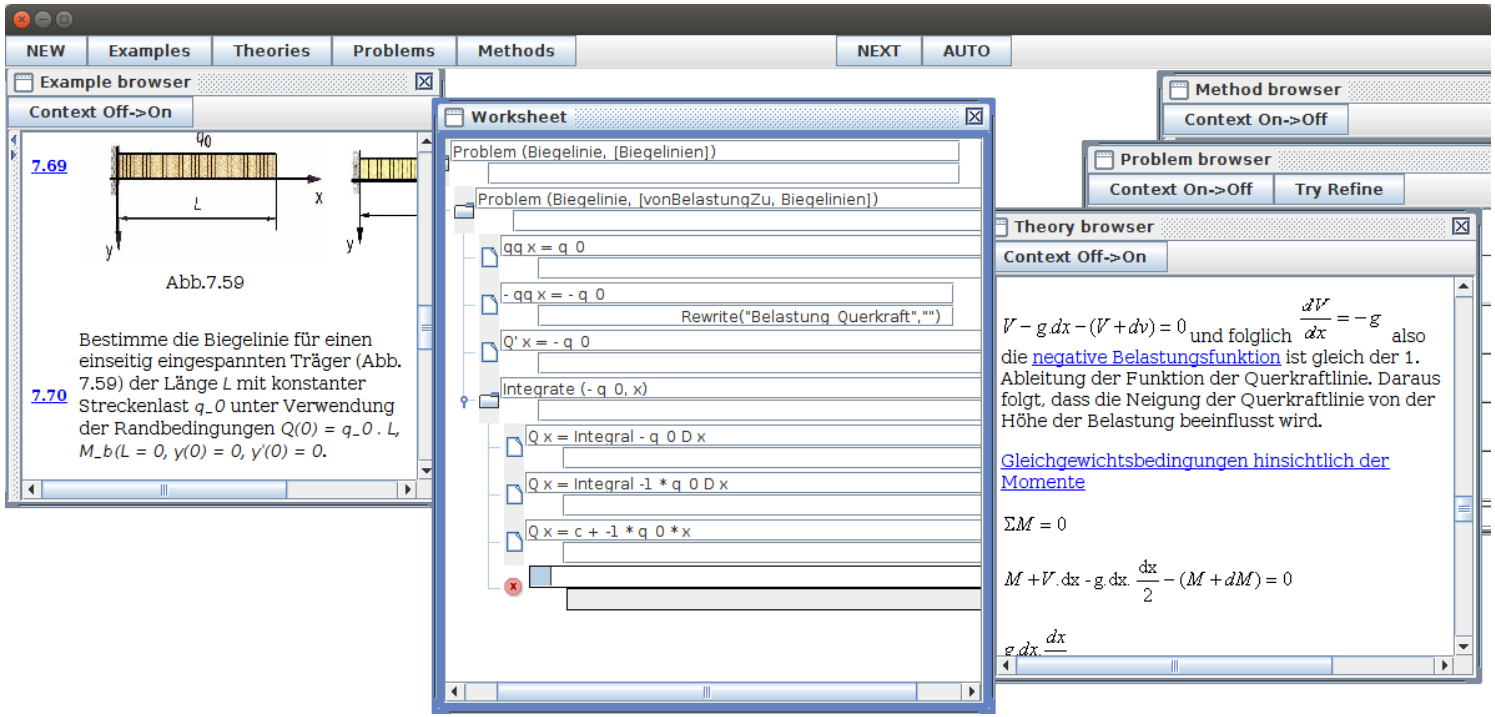


Figure: Conventional worksheet on $\mathcal{ISAC}$'s front-end

Proof Language

- Principal benefits
  - calculations in a conventional format
  - all steps of calculation in a consistent framework
  - each step is justified by theorems
  - specific steps equivalent to Computer Algebra
  - Computer Algebra decomposed into elementary steps
- Added value of implementation
  - change from survey to detail in the calculation tree (collapsing and expanding)
  - justification for any step can be inspected on demand
  - steps can be redone while trying alternative ways
  - alternatives can be tried in parallel windows

Proof Language

- Principal benefits
  - calculations in a conventional format
  - all steps of calculation in a consistent framework
  - each step is justified by theorems
  - specific steps equivalent to Computer Algebra
  - Computer Algebra decomposed into elementary steps
- Added value of implementation
  - change from survey to detail in the calculation tree (collapsing and expanding)
  - justification for any step can be inspected on demand
  - steps can be redone while trying alternative ways
  - alternatives can be tried in parallel windows

Outline

Specification Language

Formal specification of the previous **Solution:**

```
01 Problem (Biegelinie, [Biegelinien])
02   Specification:
03     Model:
04       Given     : Traegerlaenge L, Streckenlast q_0
05       Where     : q_0 ist_integrierbar_auf [0, L]
06       Find      : Biegelinie y
07       Relate    : Randbedingungen [Q 0 = q_0 · L, M_b L = 0, y 0 = 0, d/dx y 0 = 0]
08     References:
09       Theory    : Biegelinie
10     x Problem   : ["Biegelinien"]
11     o Method    : ["IntegrierenUndKonstanteBestimmen2"]
12   Solution:
```

Hidden data for *“next step guidance”*:

[([*Traegerlaenge L, Streckenlast* q_0*, Biegelinie y,*
Randbedingungen [$Q\,0 = q_0 \cdot L$, $M_b\,L = 0$, $y\,0 = 0$, $\frac{d}{dx}\,y\,0 = 0$]*, FunktionsVariable x*]
(*"Biegelinie",* [*"Biegelinien"*]*,* [*"IntegrierenUndKonstanteBestimmen2"*]))]

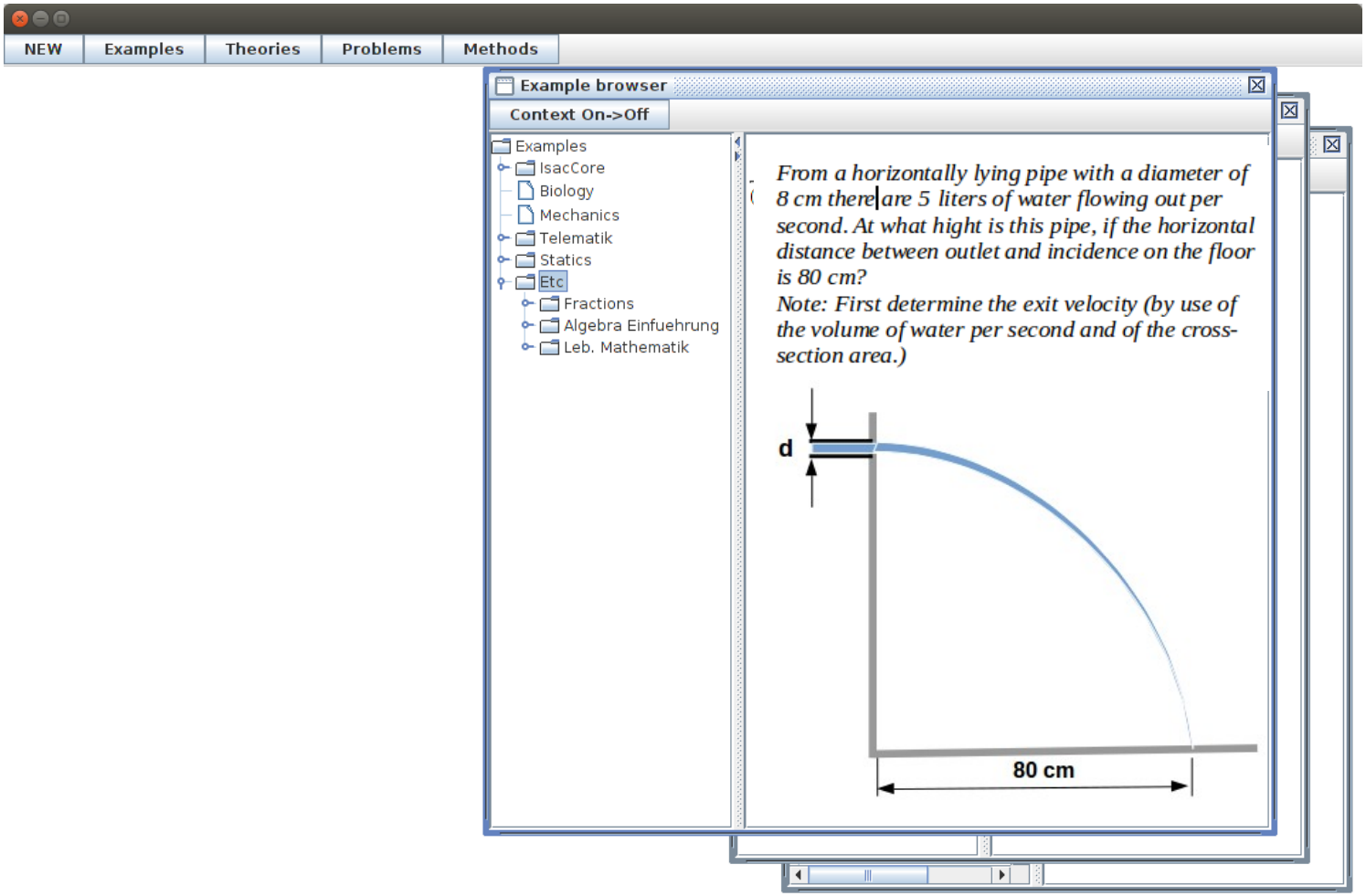
Specification Language

- Principal benefits
  - formal specification prepares mechanical solution
  - pre-condition determines solvability
  - post-condition makes essence of a problem explicit
  - problems decomposed into sub-problems (with specifications)
- Added value of implementation
  - specifications can be easily searched and tried
  - trees of specifications allow automated refinement
  - successfully specified problems solved by key stroke
  - sub-problems can be interactively arranged
  - specifications as black boxes raises abstraction in problem solving, see slide movie

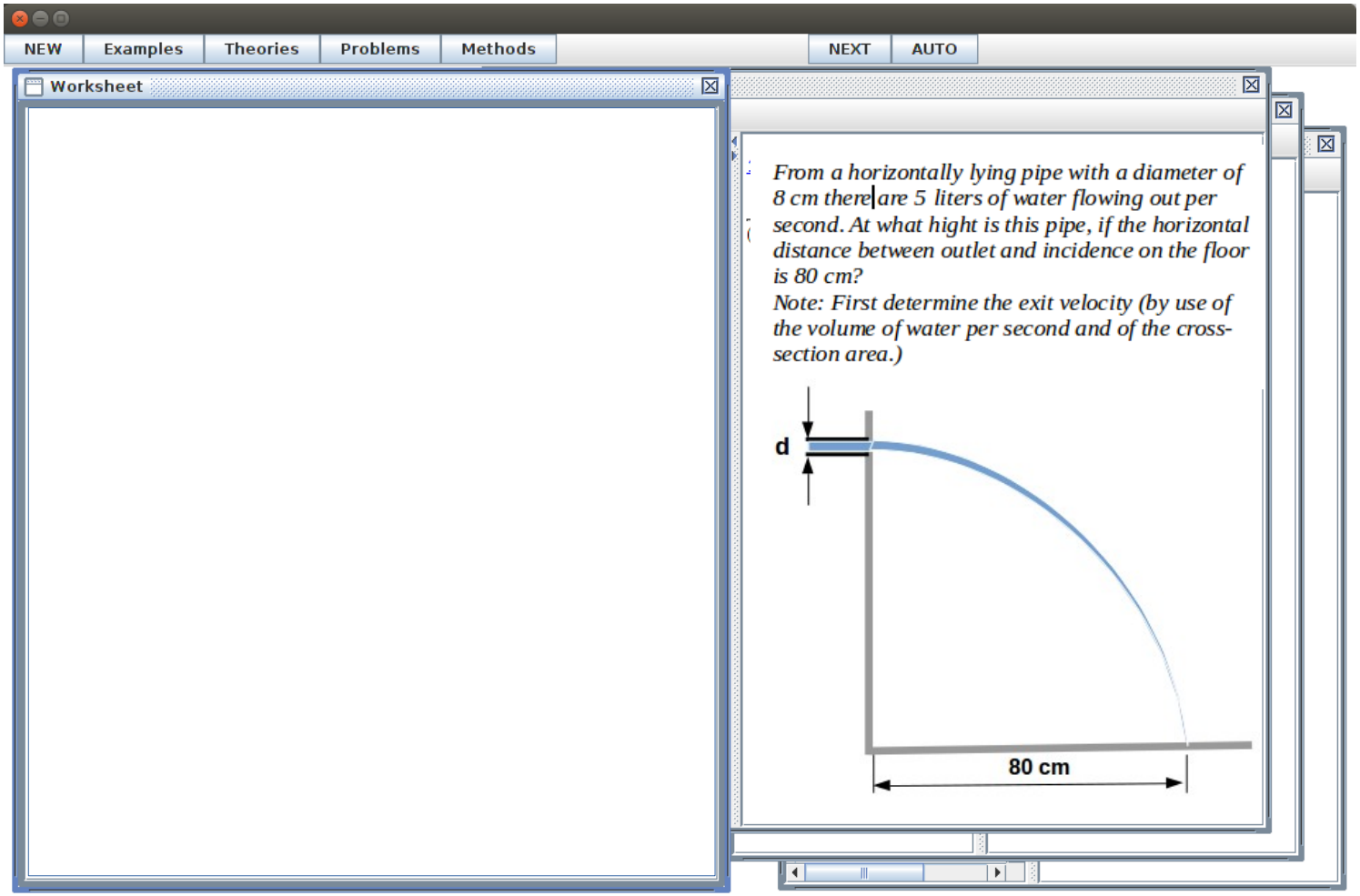
Specification Language

- Principal benefits
  - formal specification prepares mechanical solution
  - pre-condition determines solvability
  - post-condition makes essence of a problem explicit
  - problems decomposed into sub-problems (with specifications)
- Added value of implementation
  - specifications can be easily searched and tried
  - trees of specifications allow automated refinement
  - successfully specified problems solved by key stroke
  - sub-problems can be interactively arranged
  - specifications as black boxes raises abstraction in problem solving, see slide movie

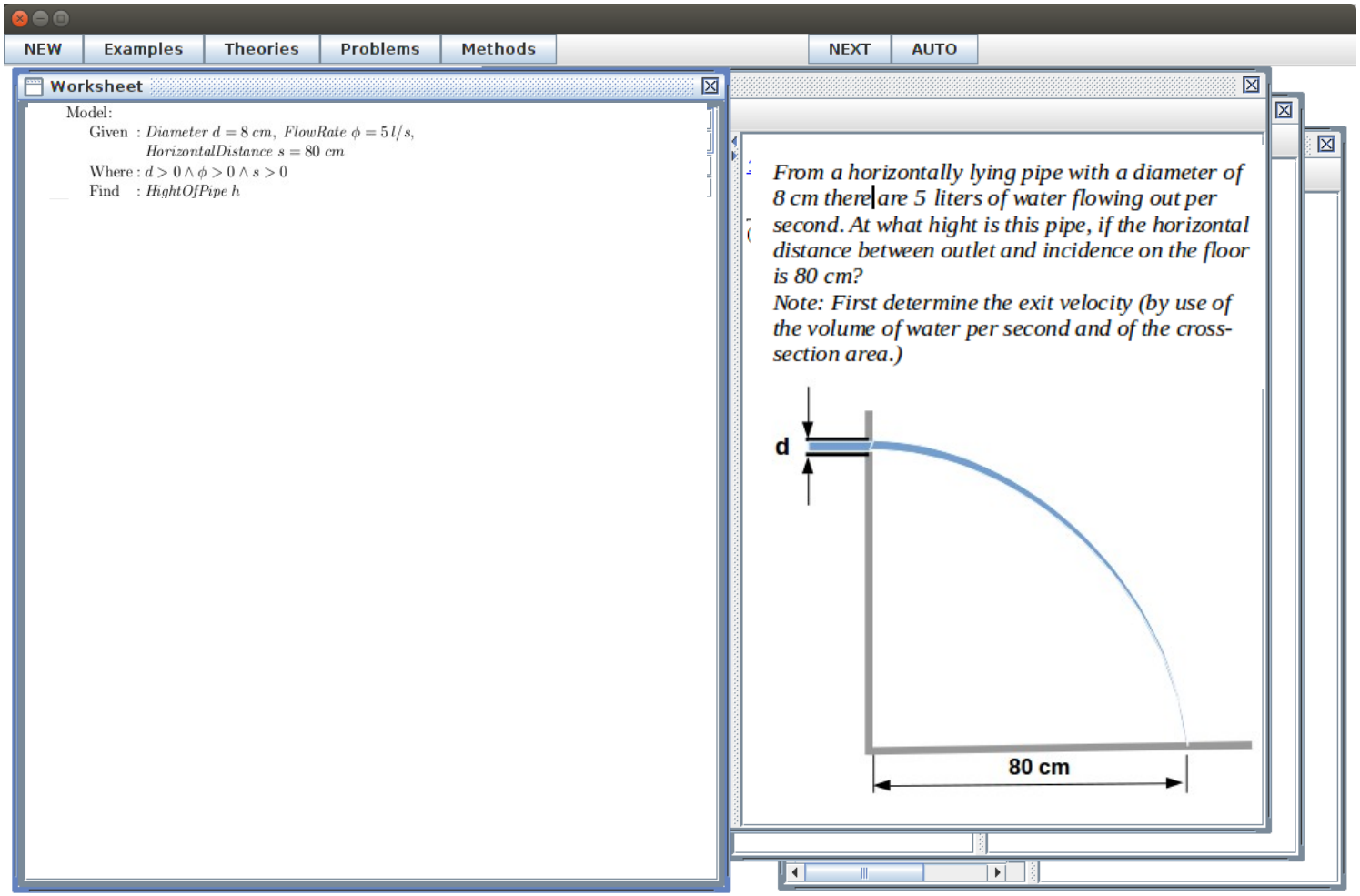
Start Example



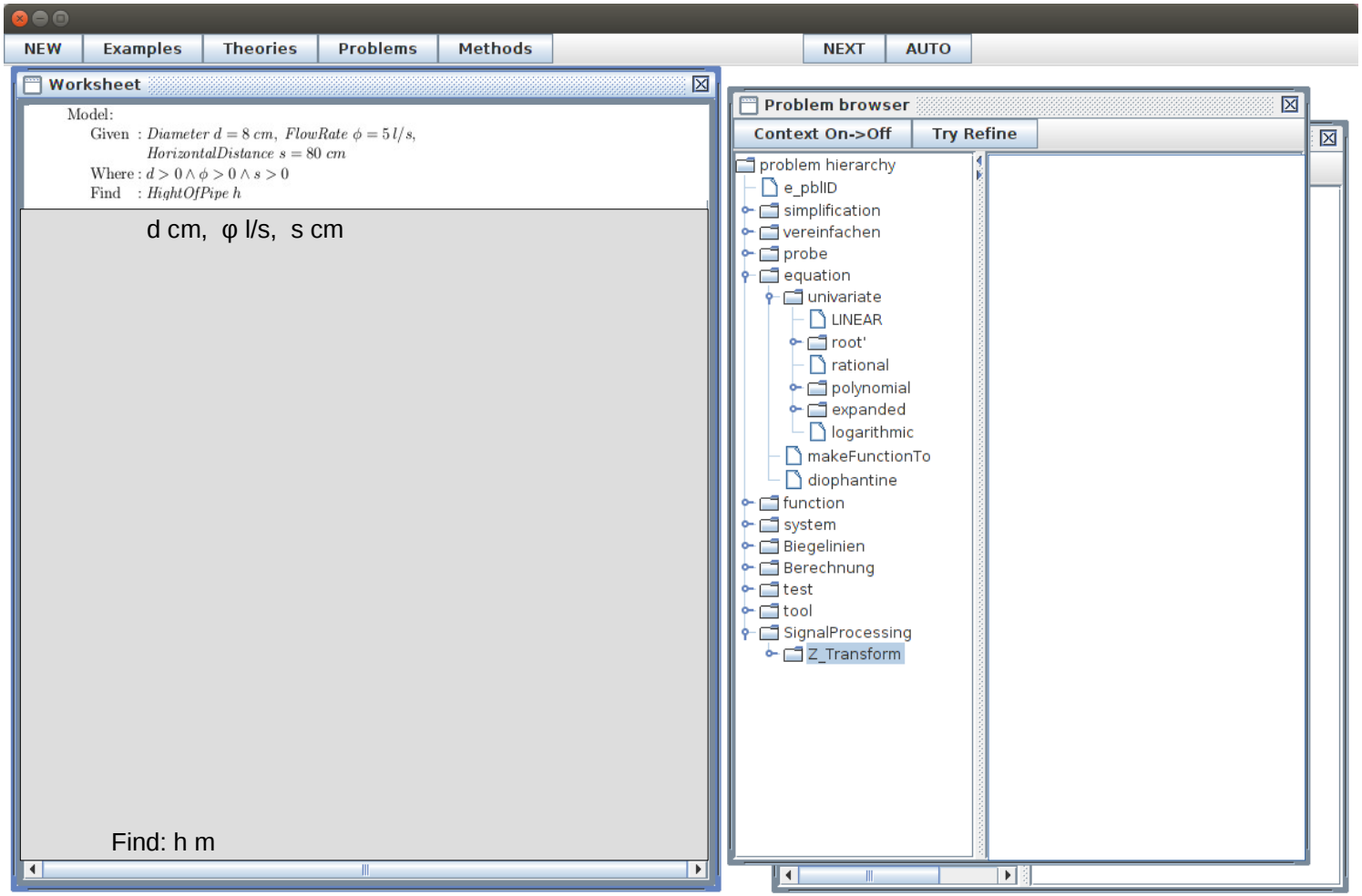
Start Example



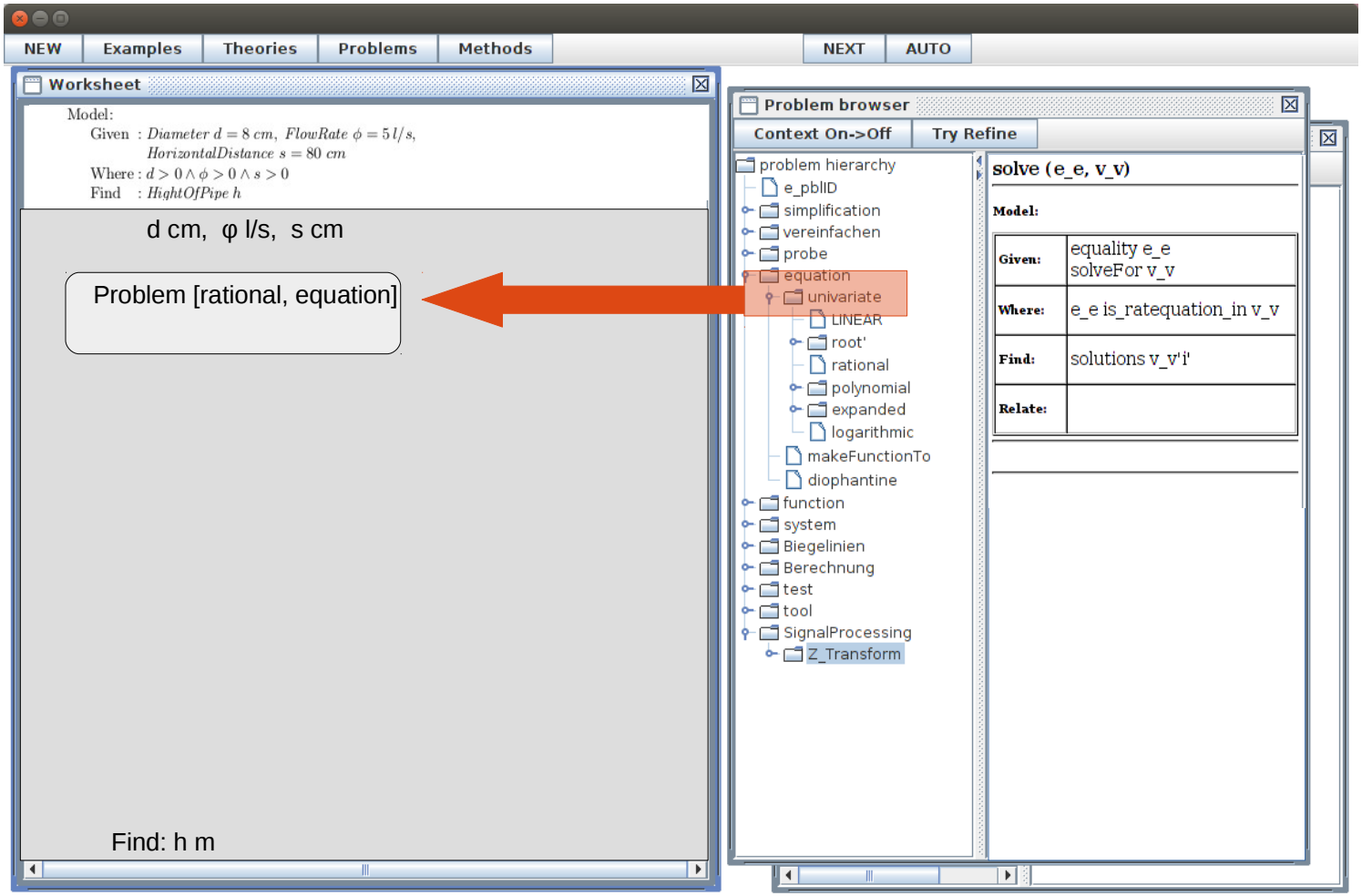
Problem modelled ok



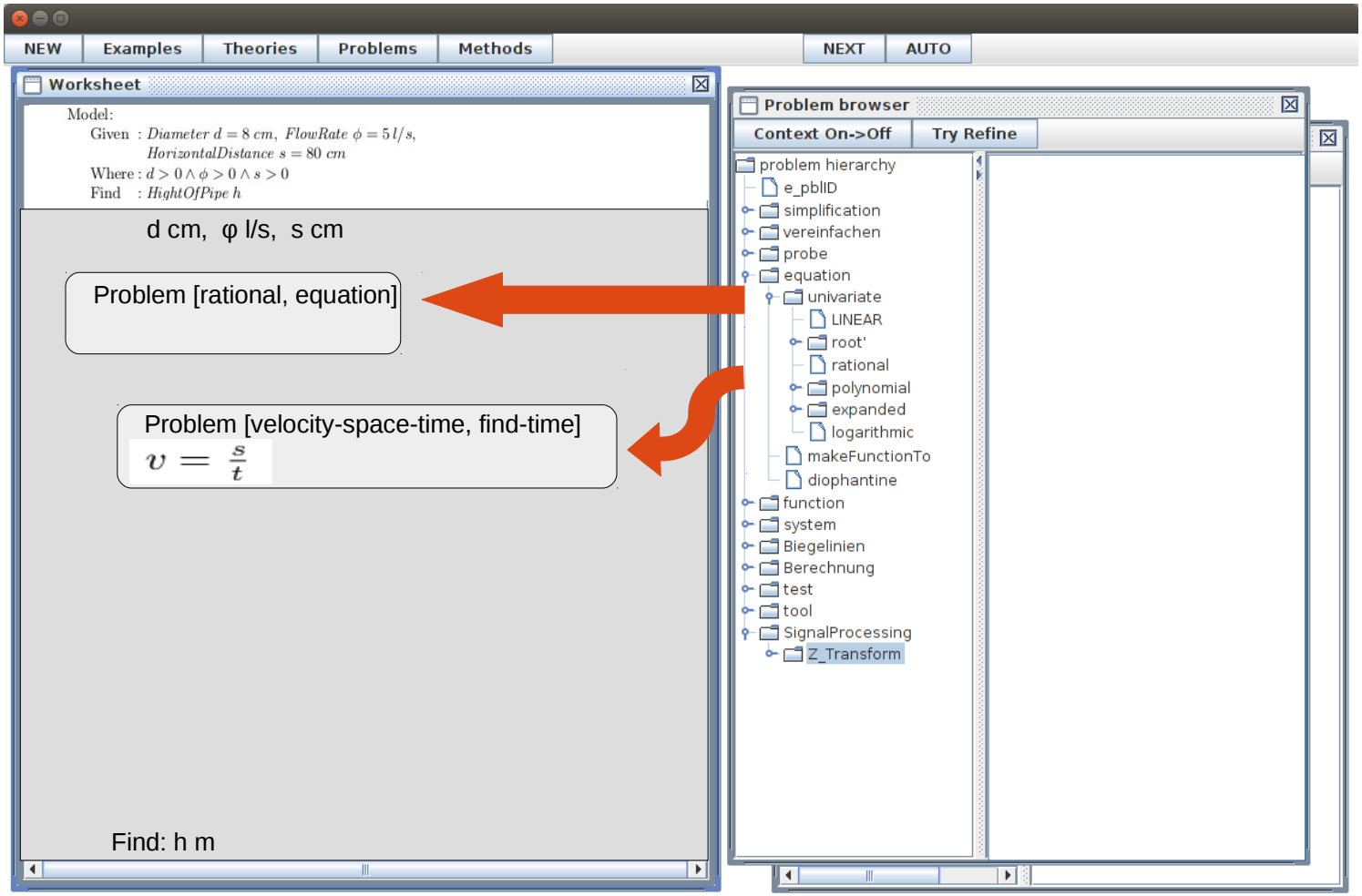
Start considering sub-problems



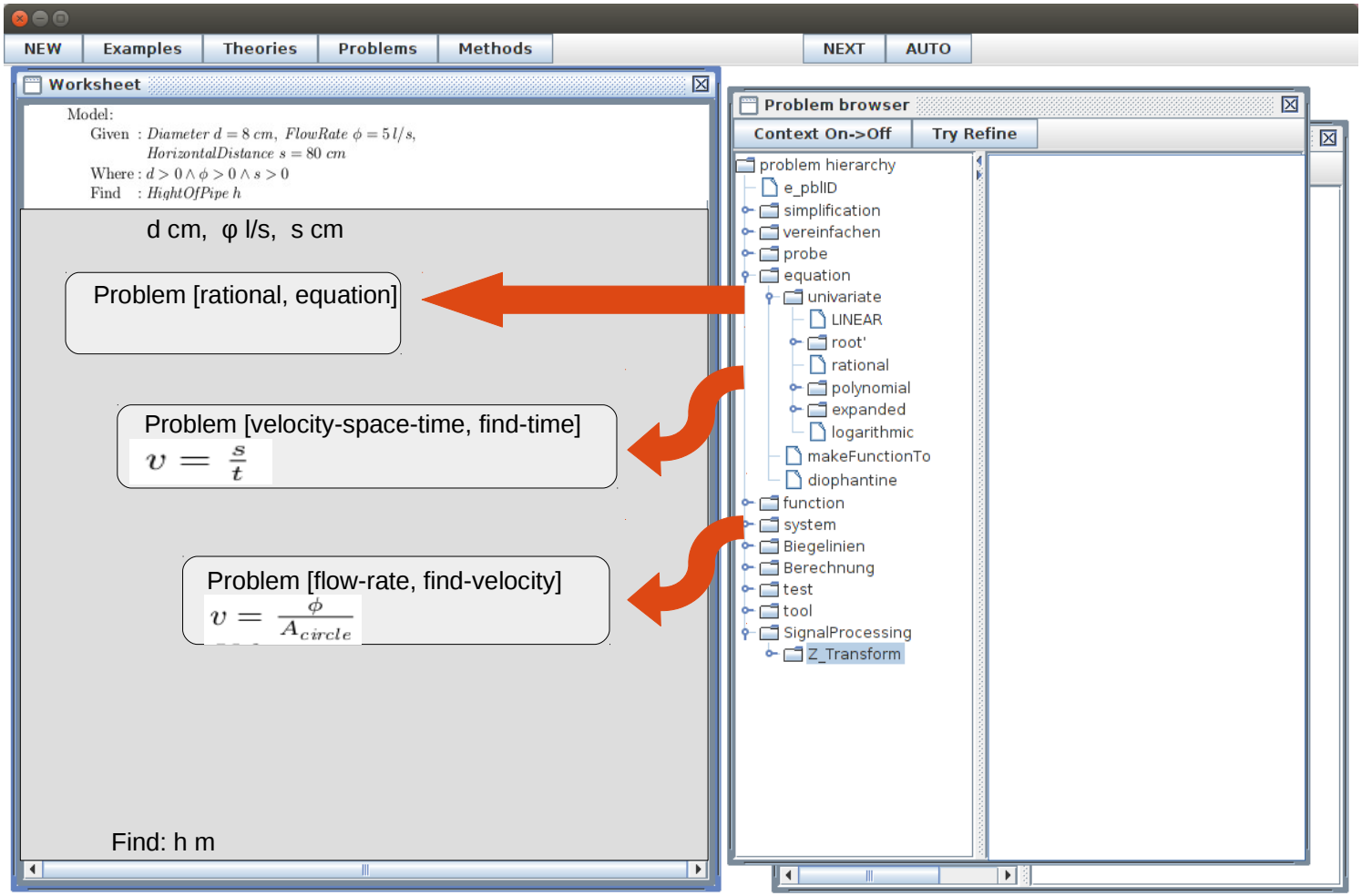
Select sub-problems



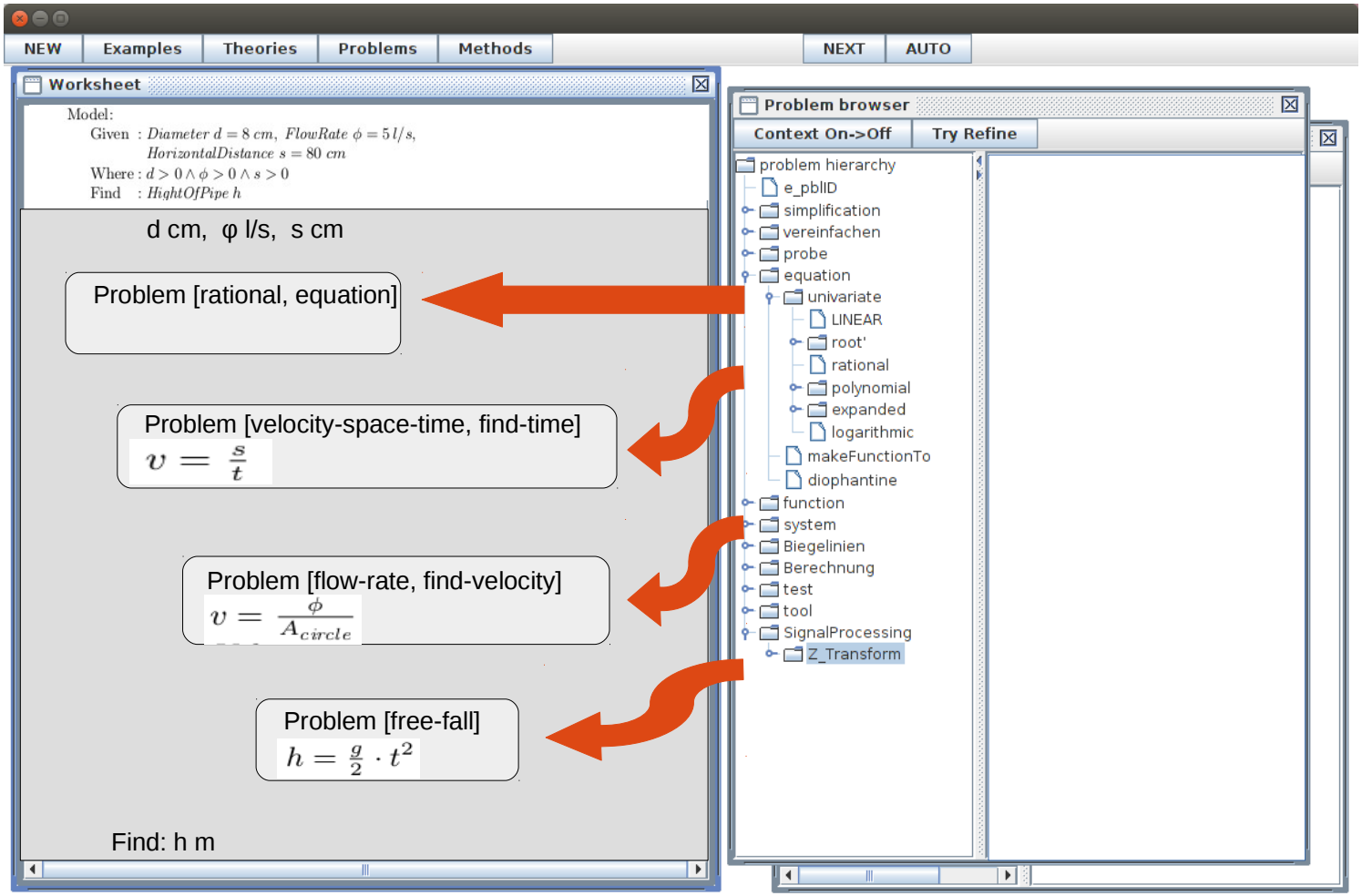
Select sub-problems



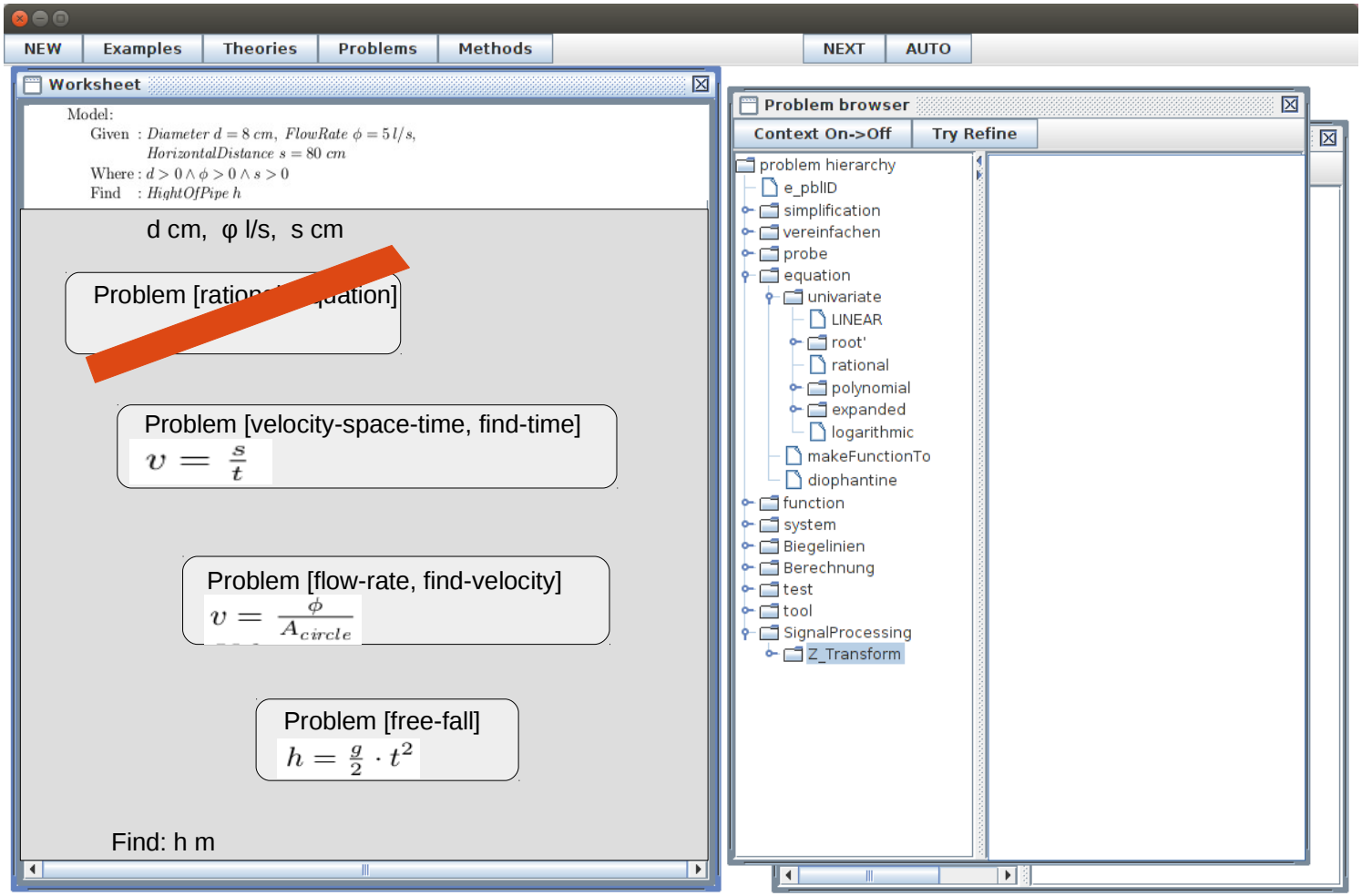
Select sub-problems



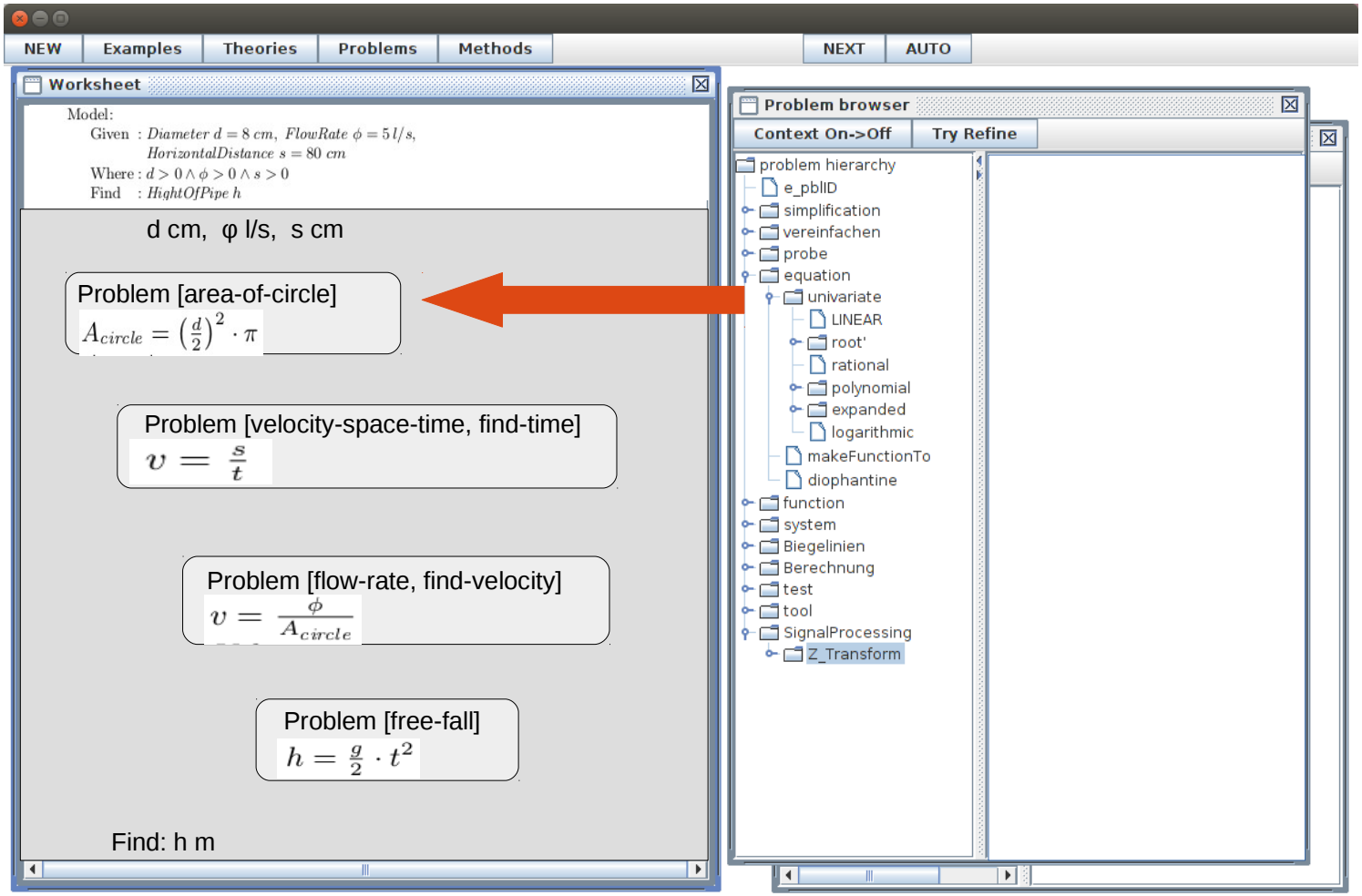
Select sub-problems



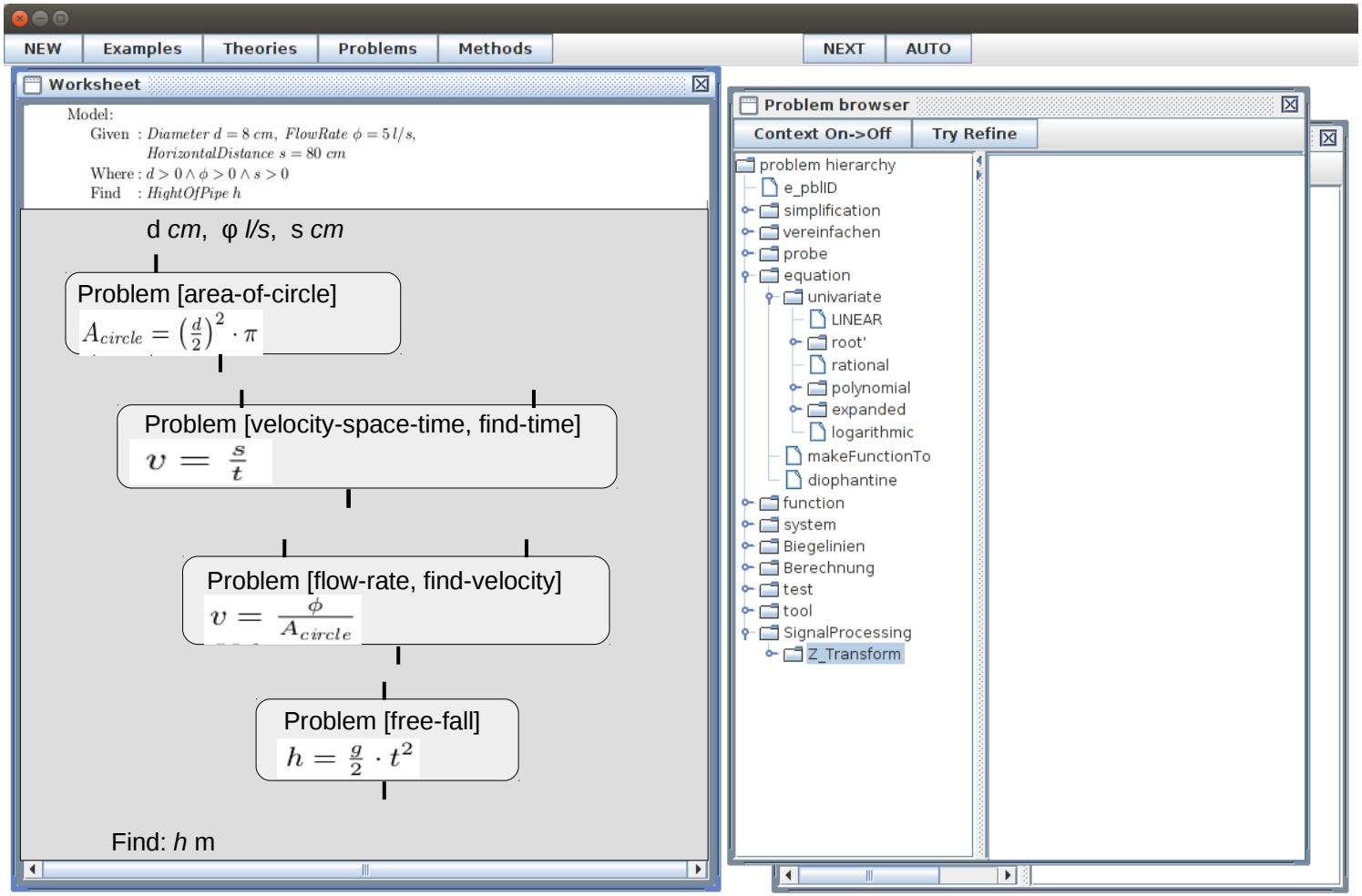
Delete irrelevant sub-problems



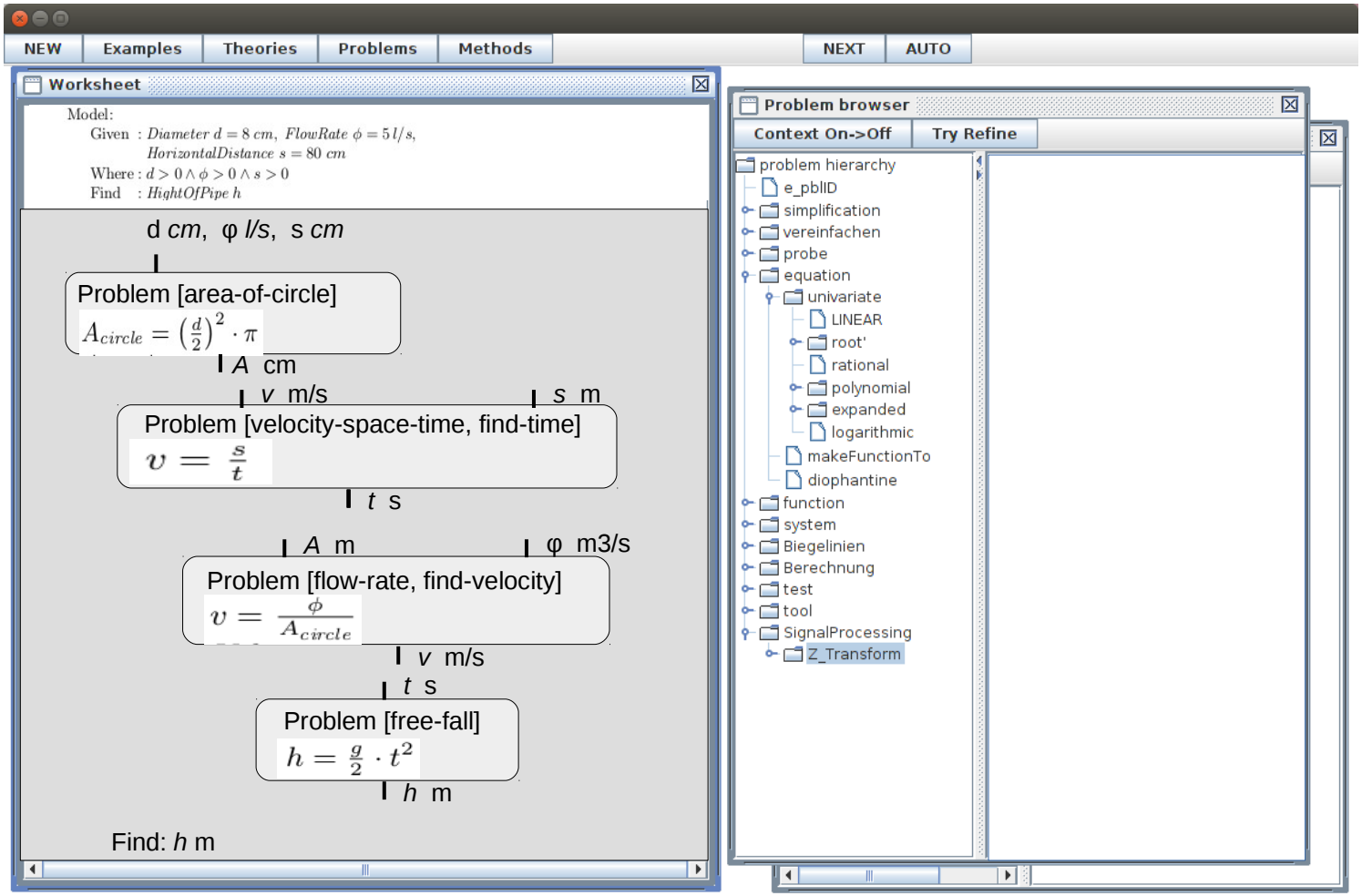
Select relevant sub-problems



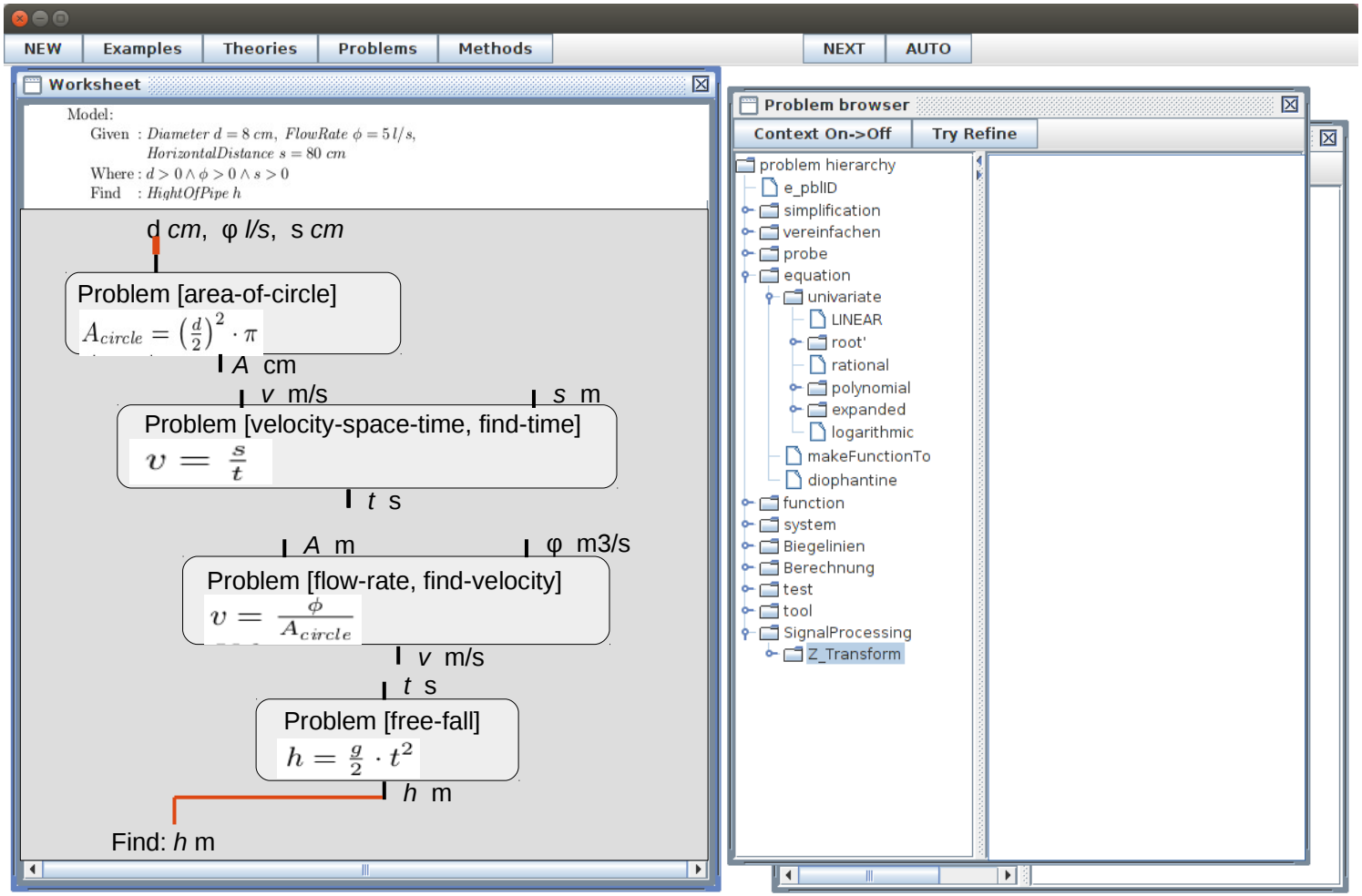
What is given / has to be found?



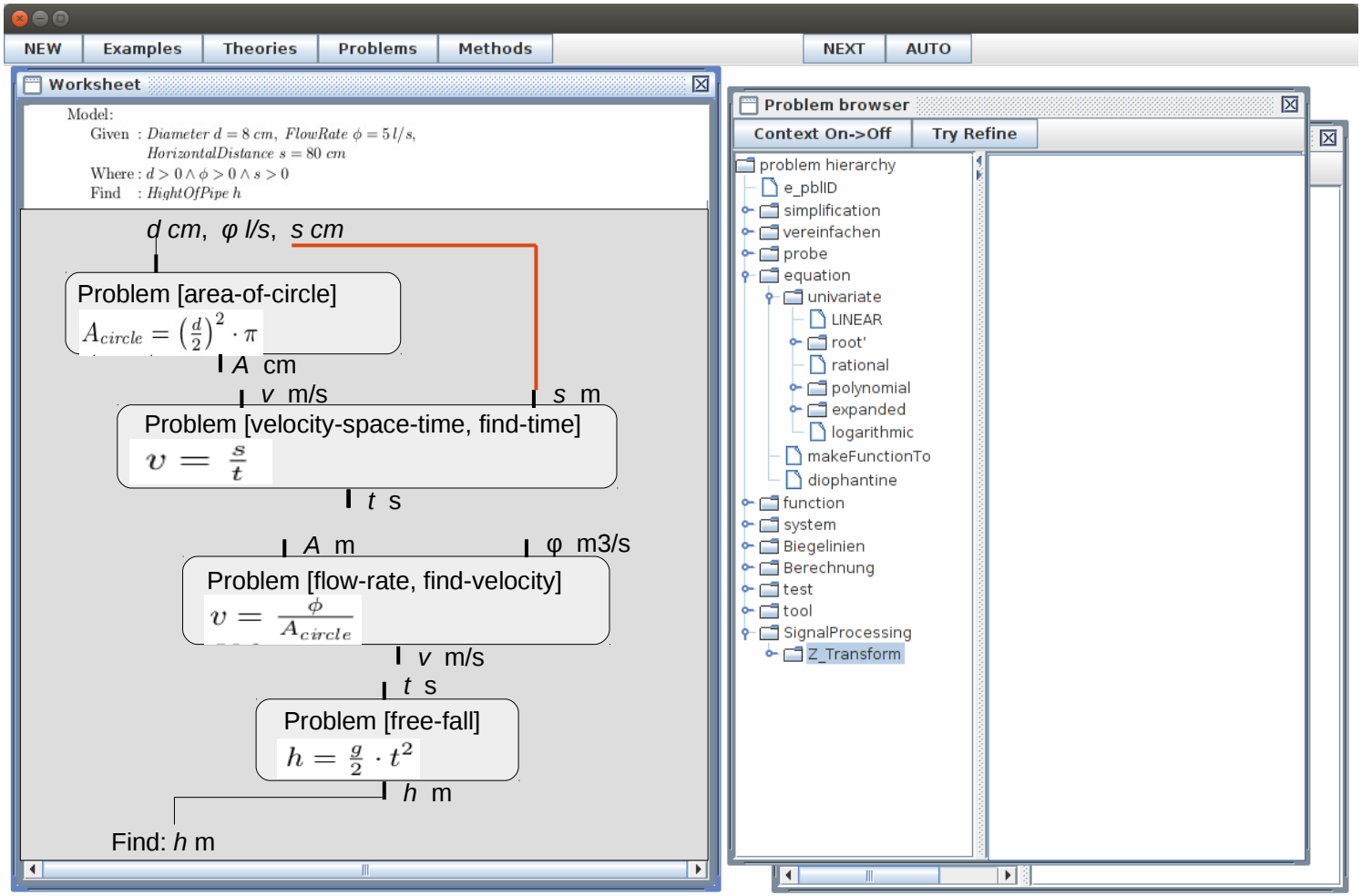
What is given / has to be found?



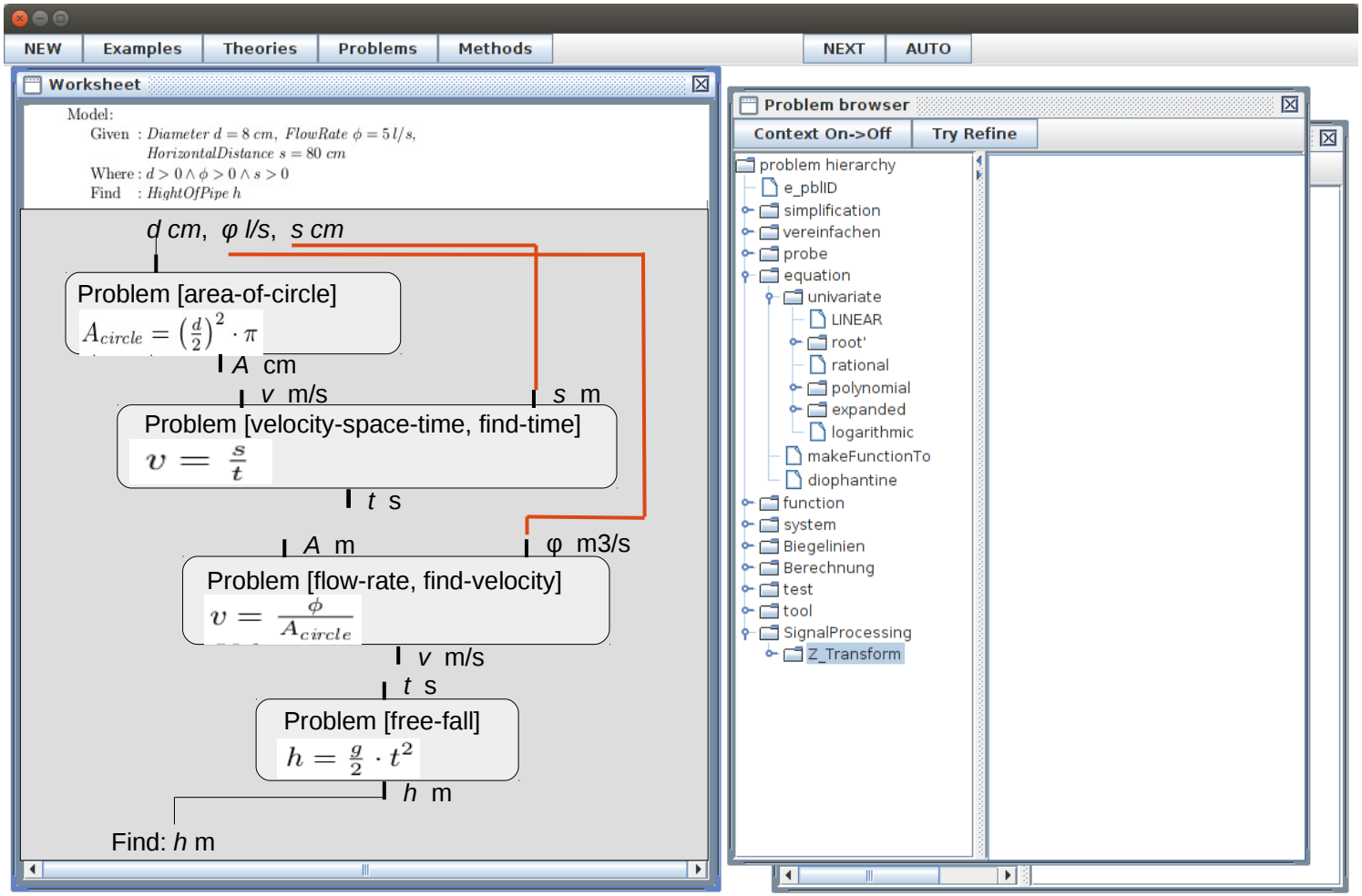
Connect “Given” and “Find”



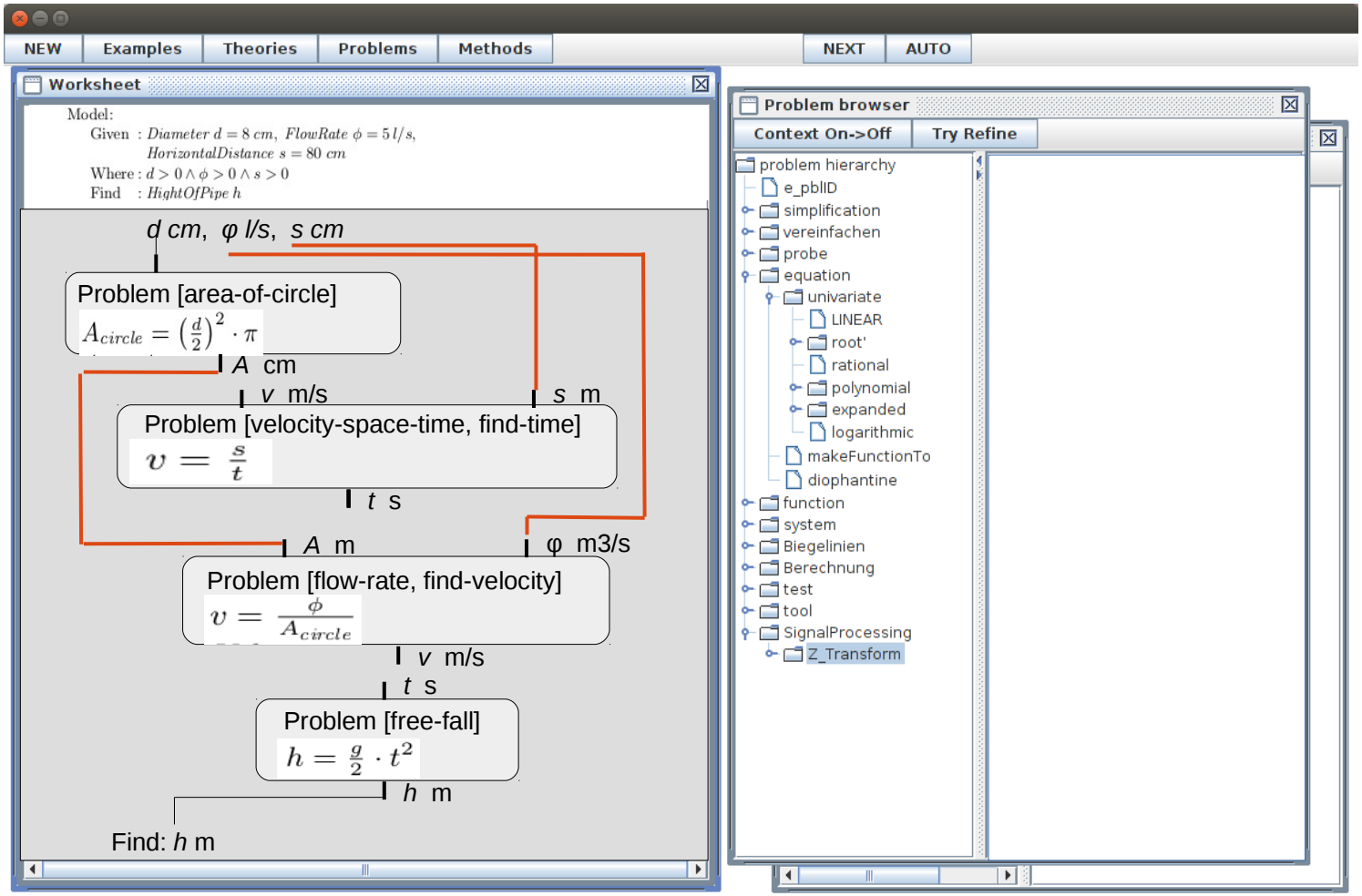
Connect “Given” and “Find”



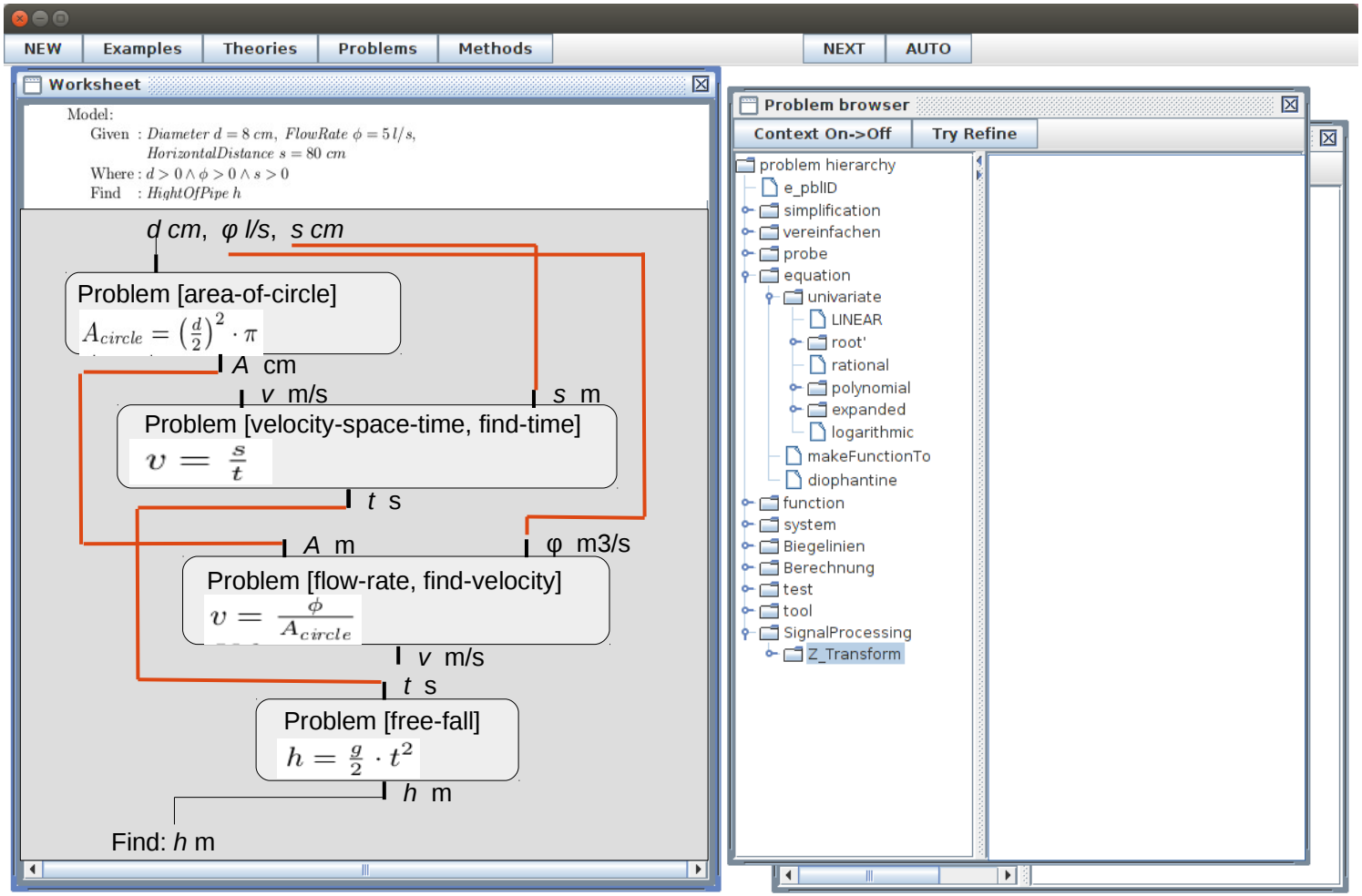
Connect “Given” and “Find”



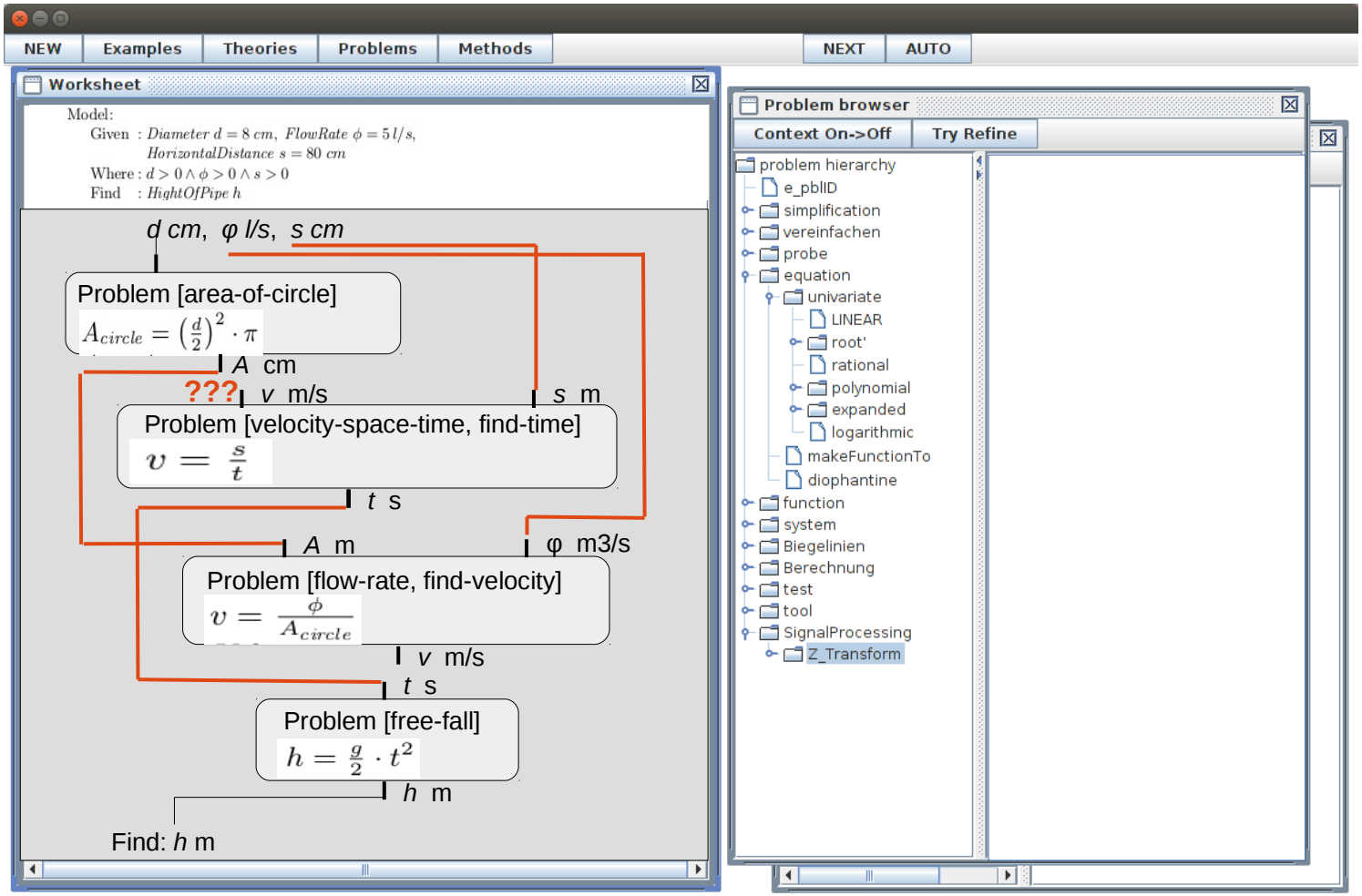
Connect “Given” and “Find”



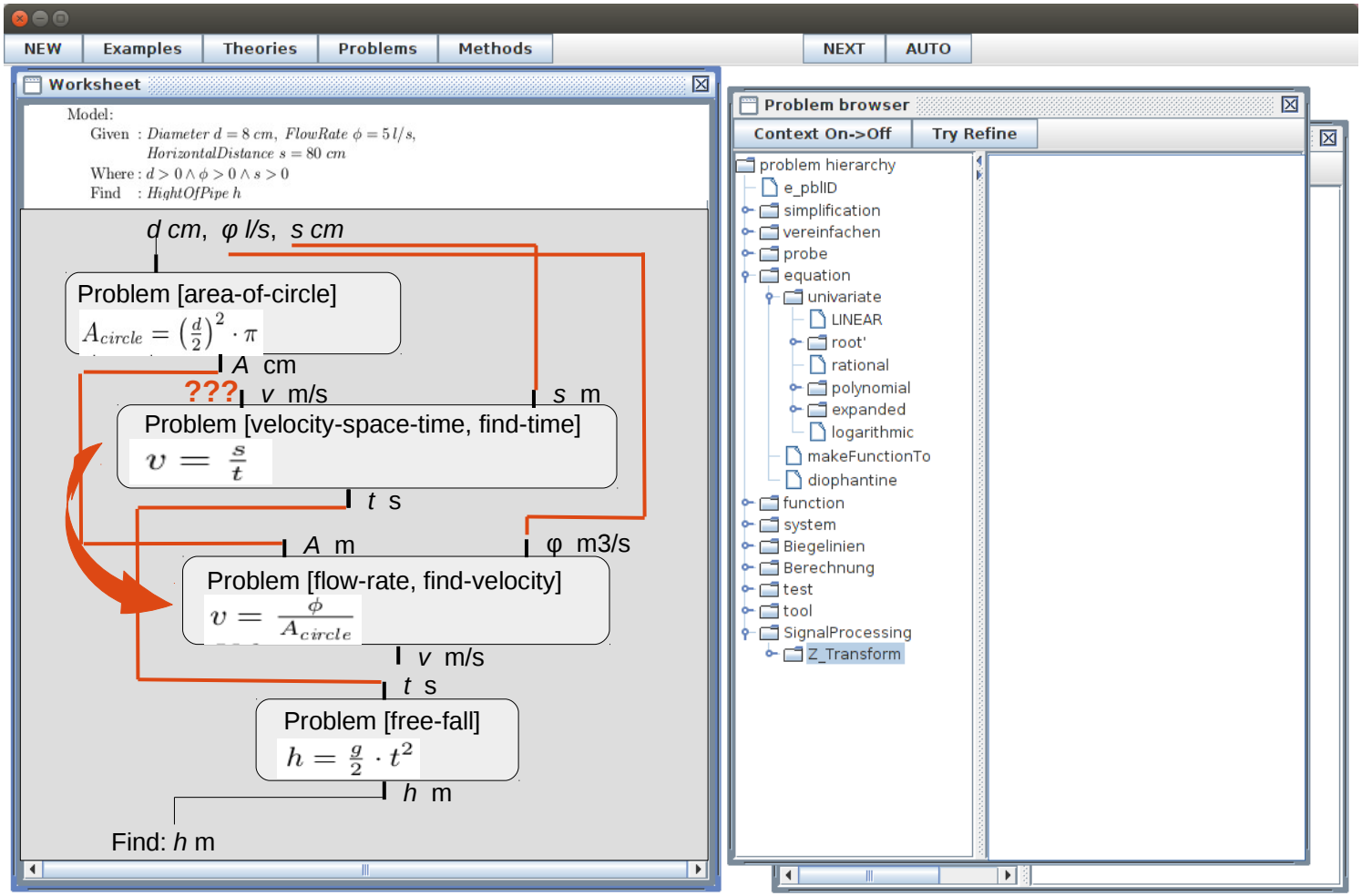
Connect “Given” and “Find”



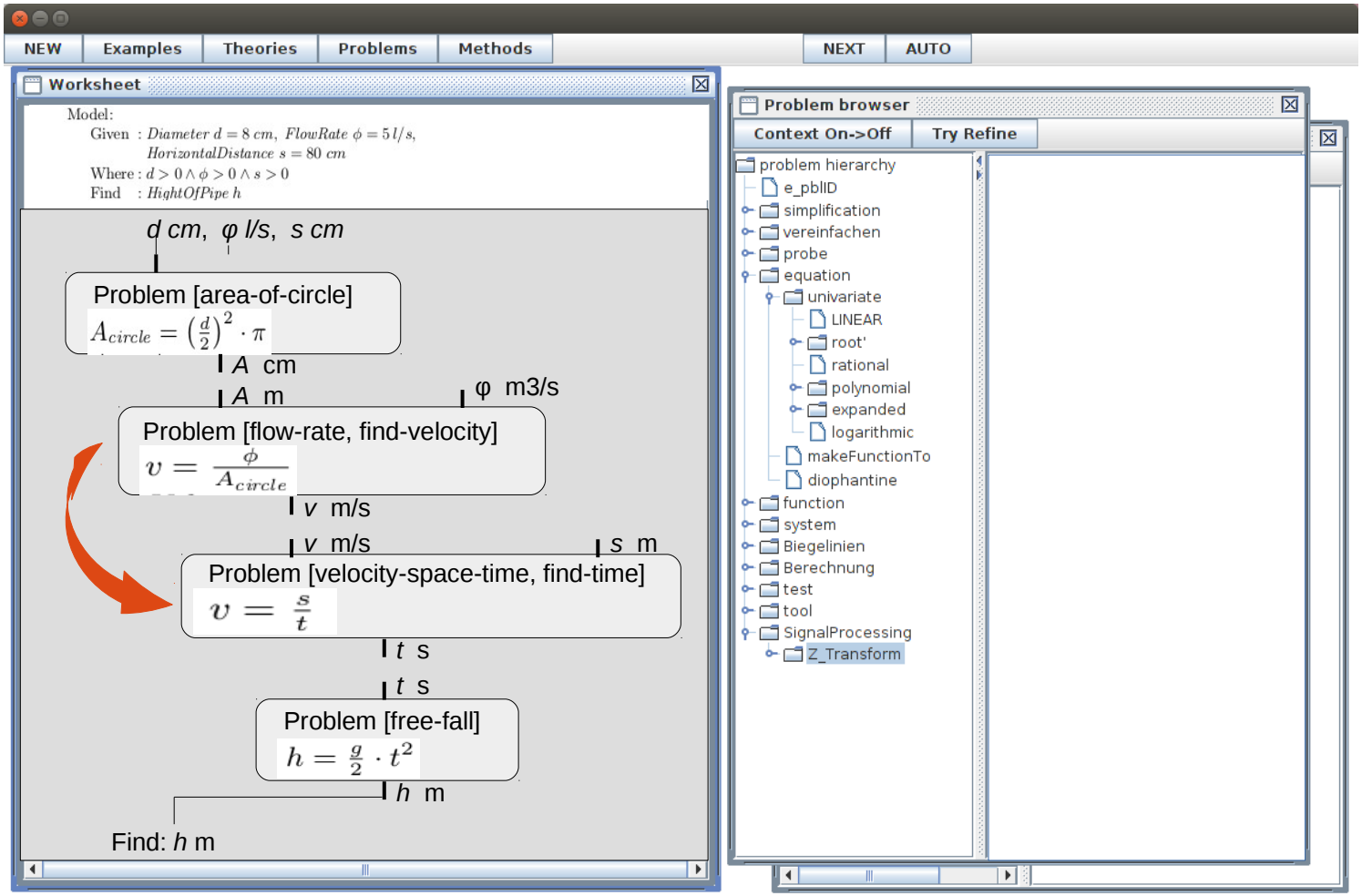
Dangling connection ???



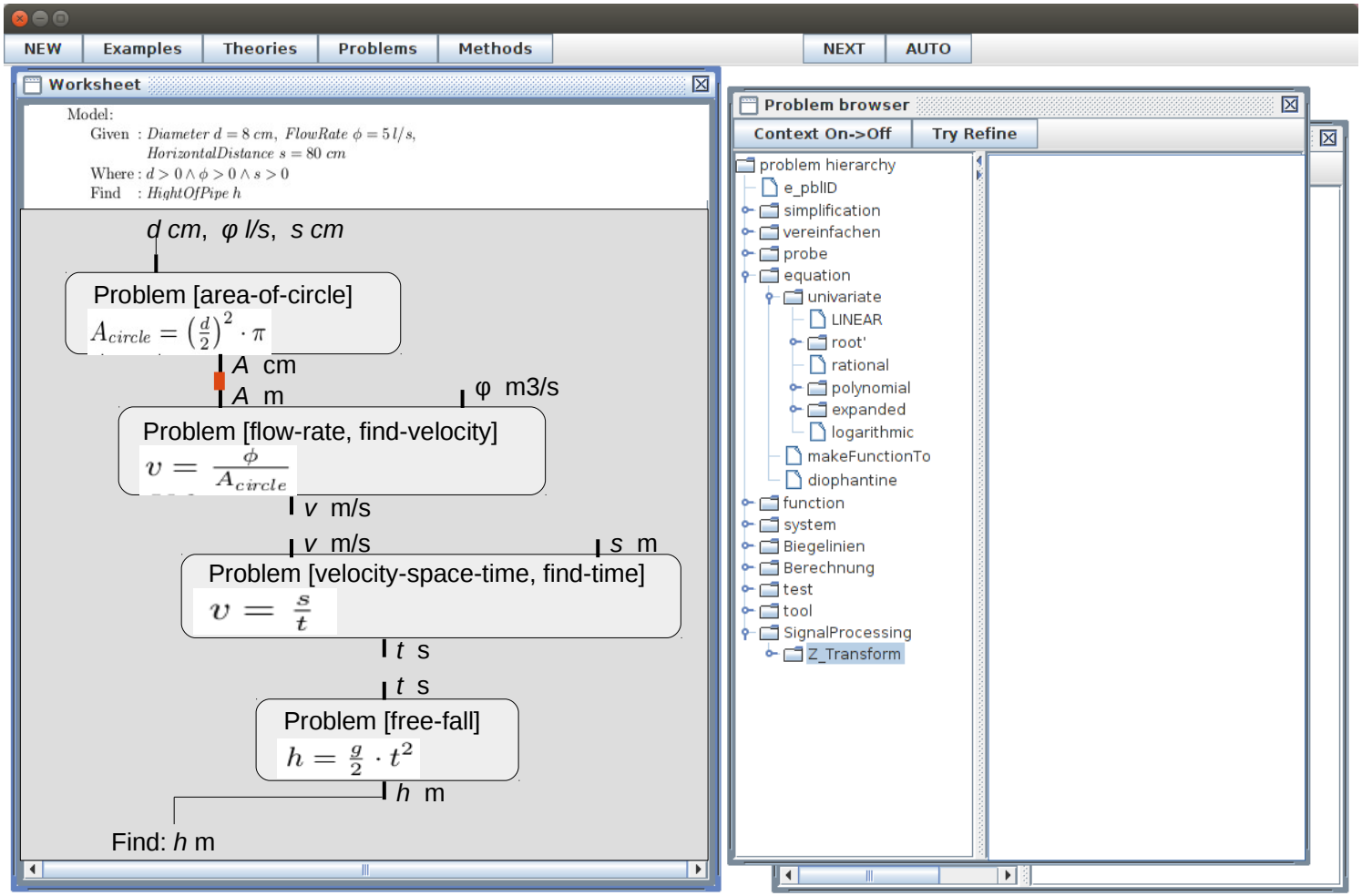
Rearrange sub-problems



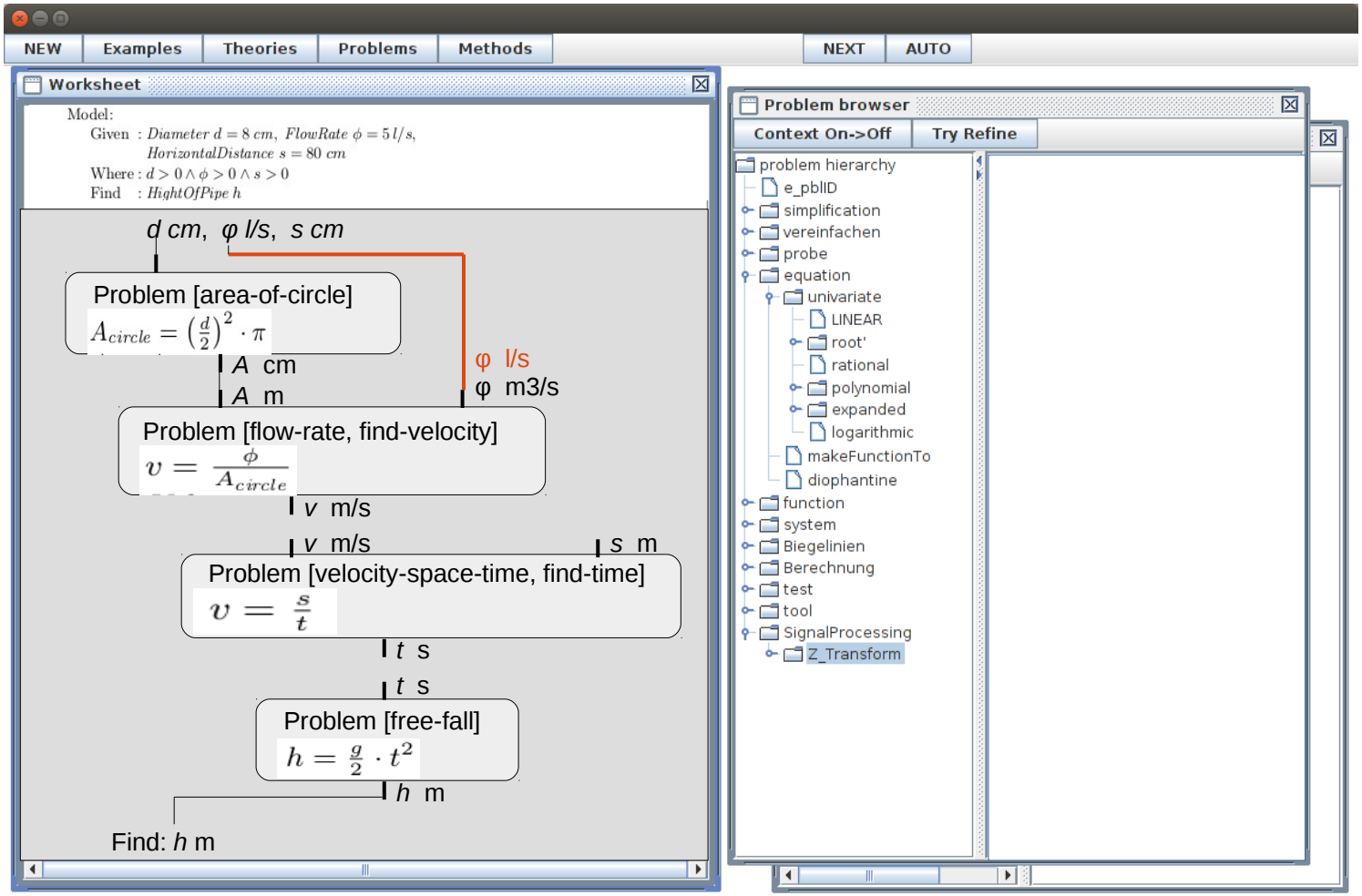
Flipped two sub-problems



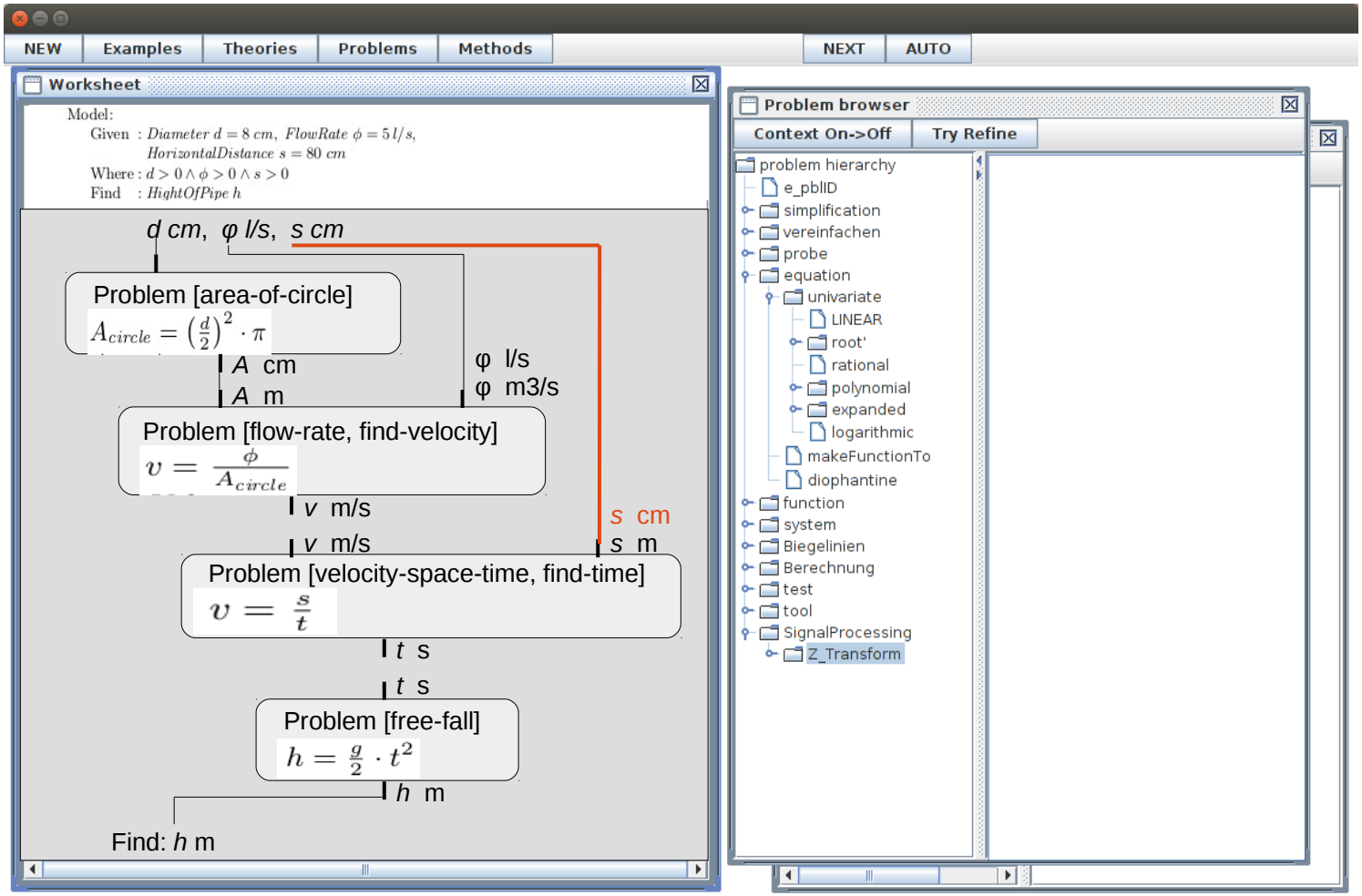
Connect “Given” and “Find”



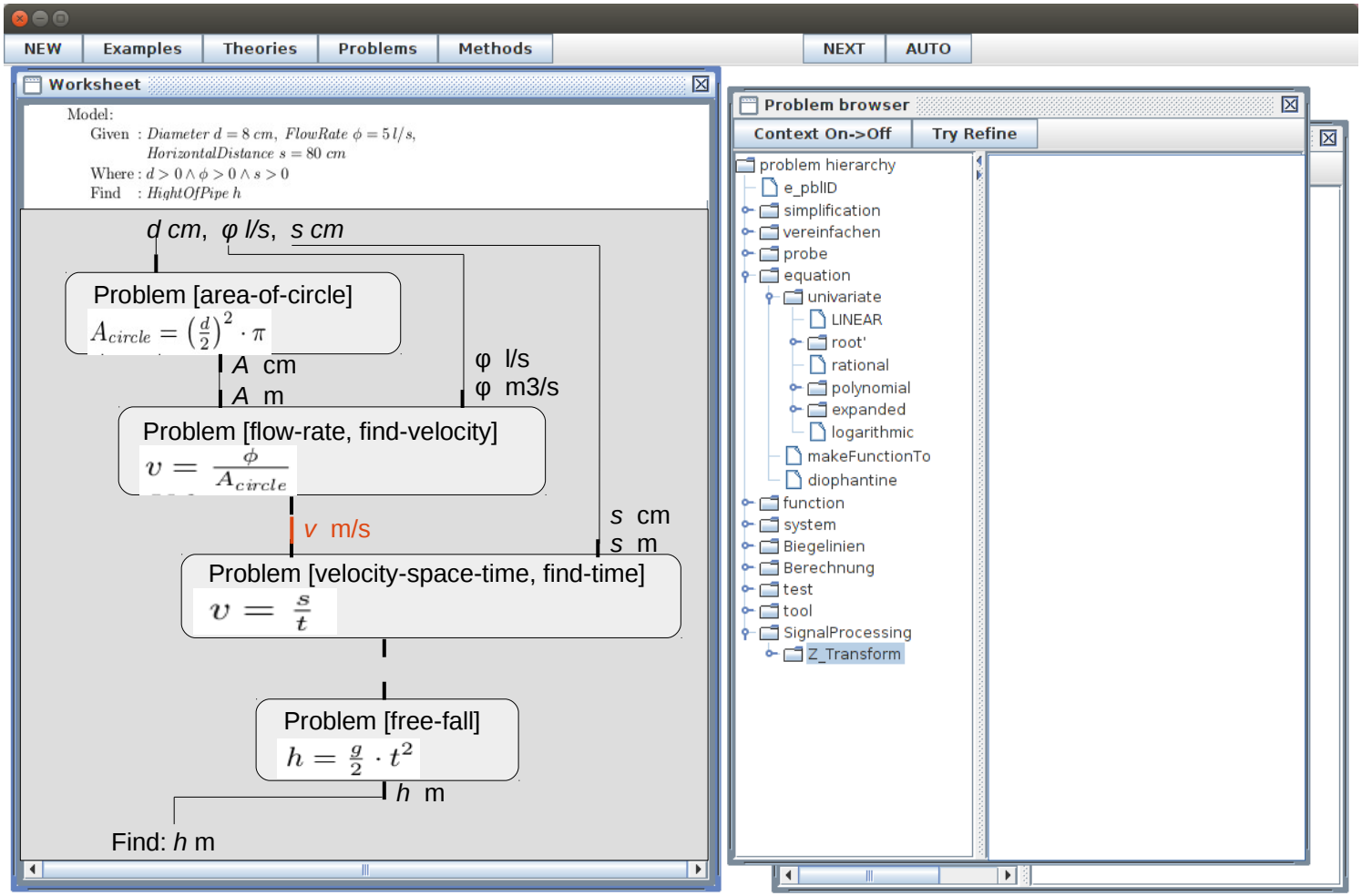
Connect “Given” and “Find”



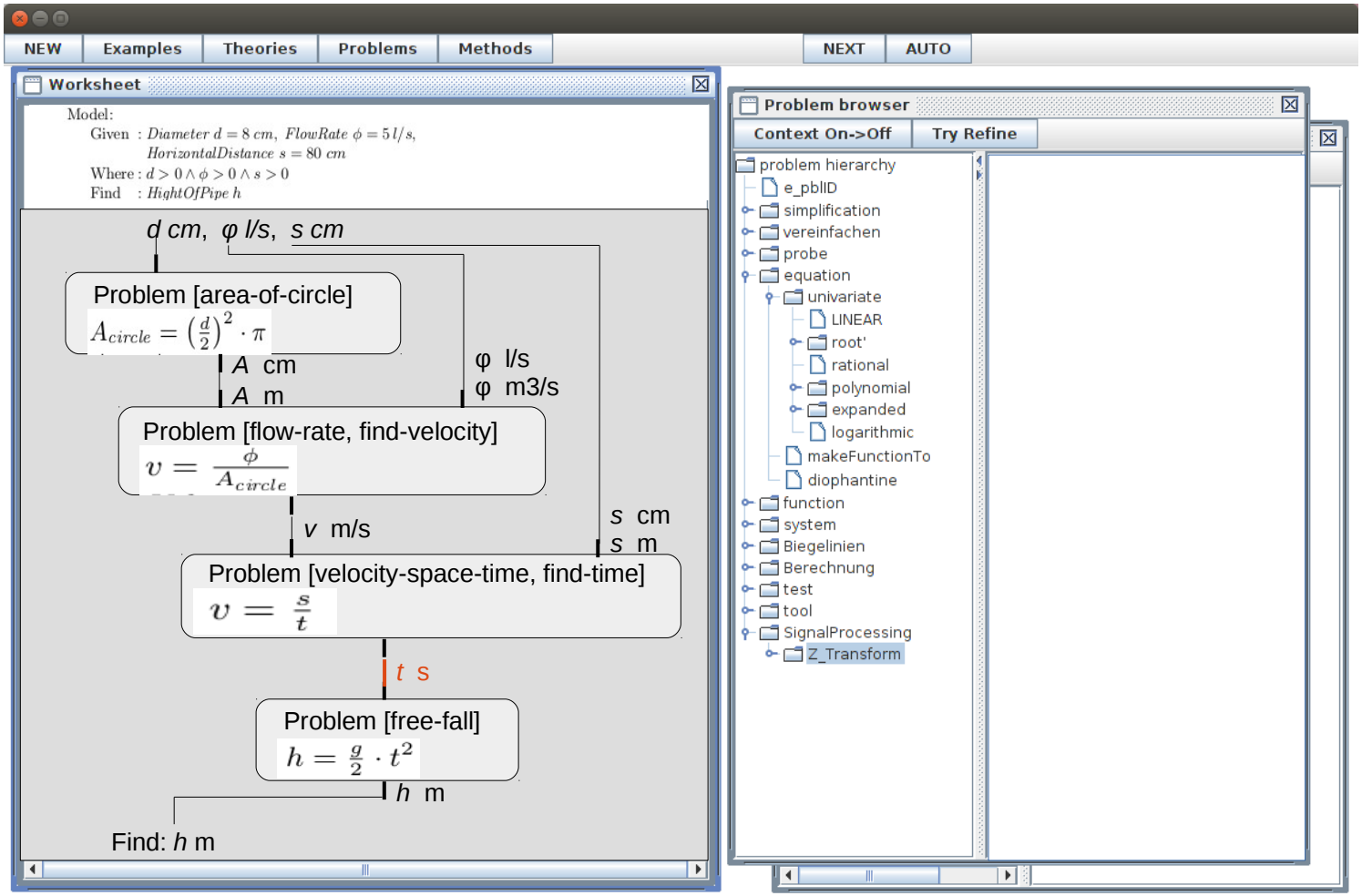
Connect “Given” and “Find”



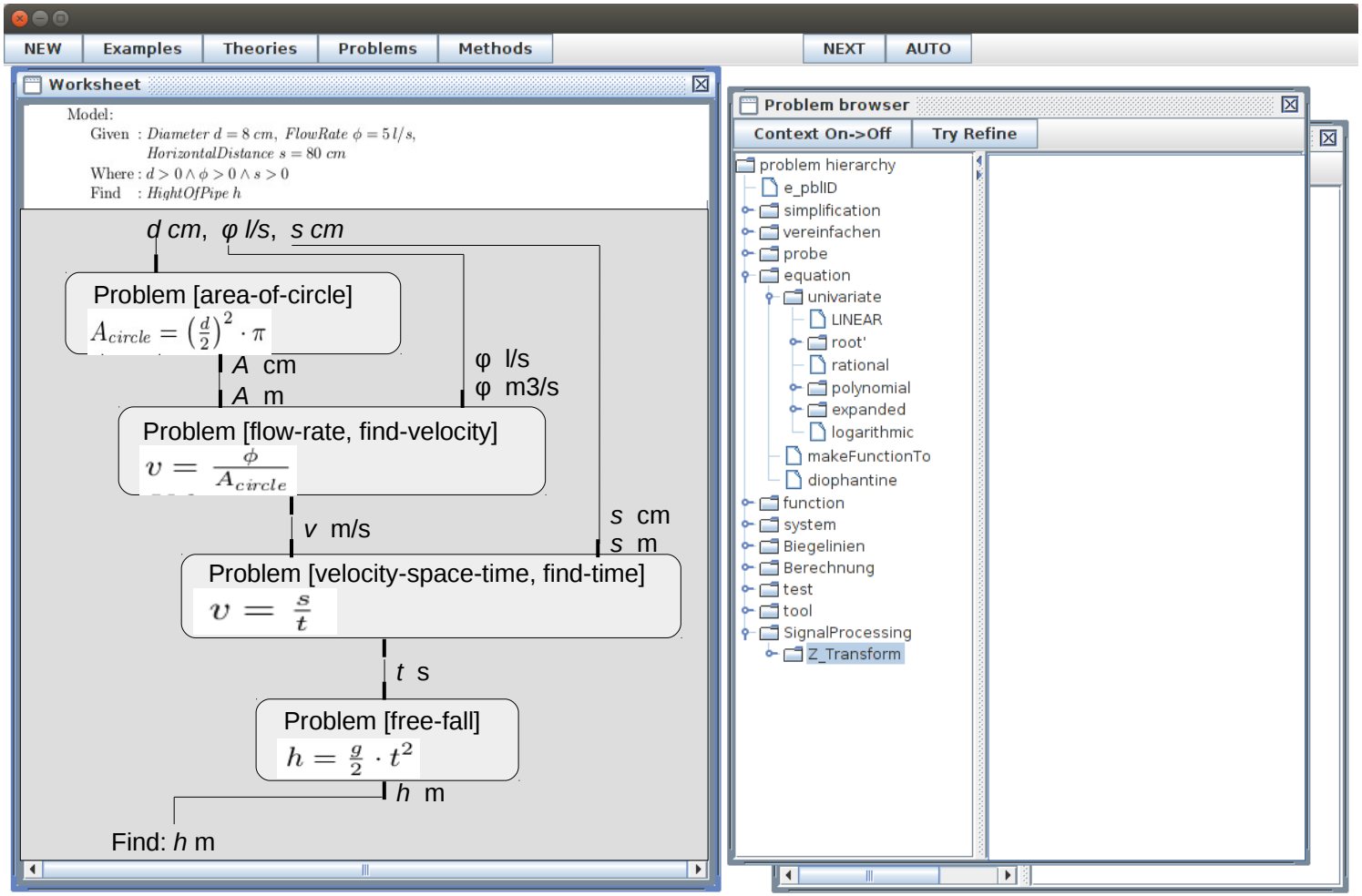
Connect “Given” and “Find”



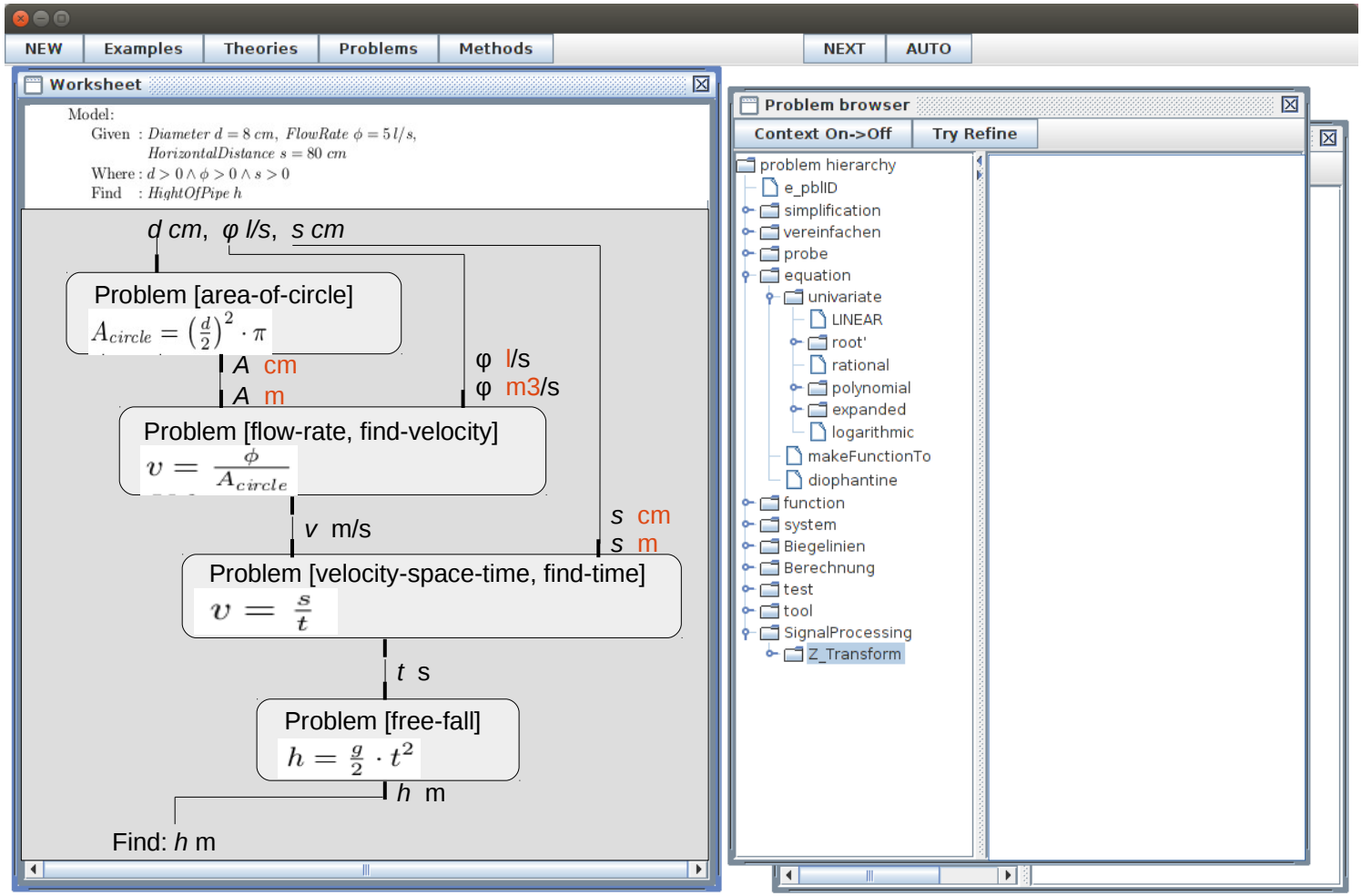
Connect “Given” and “Find”



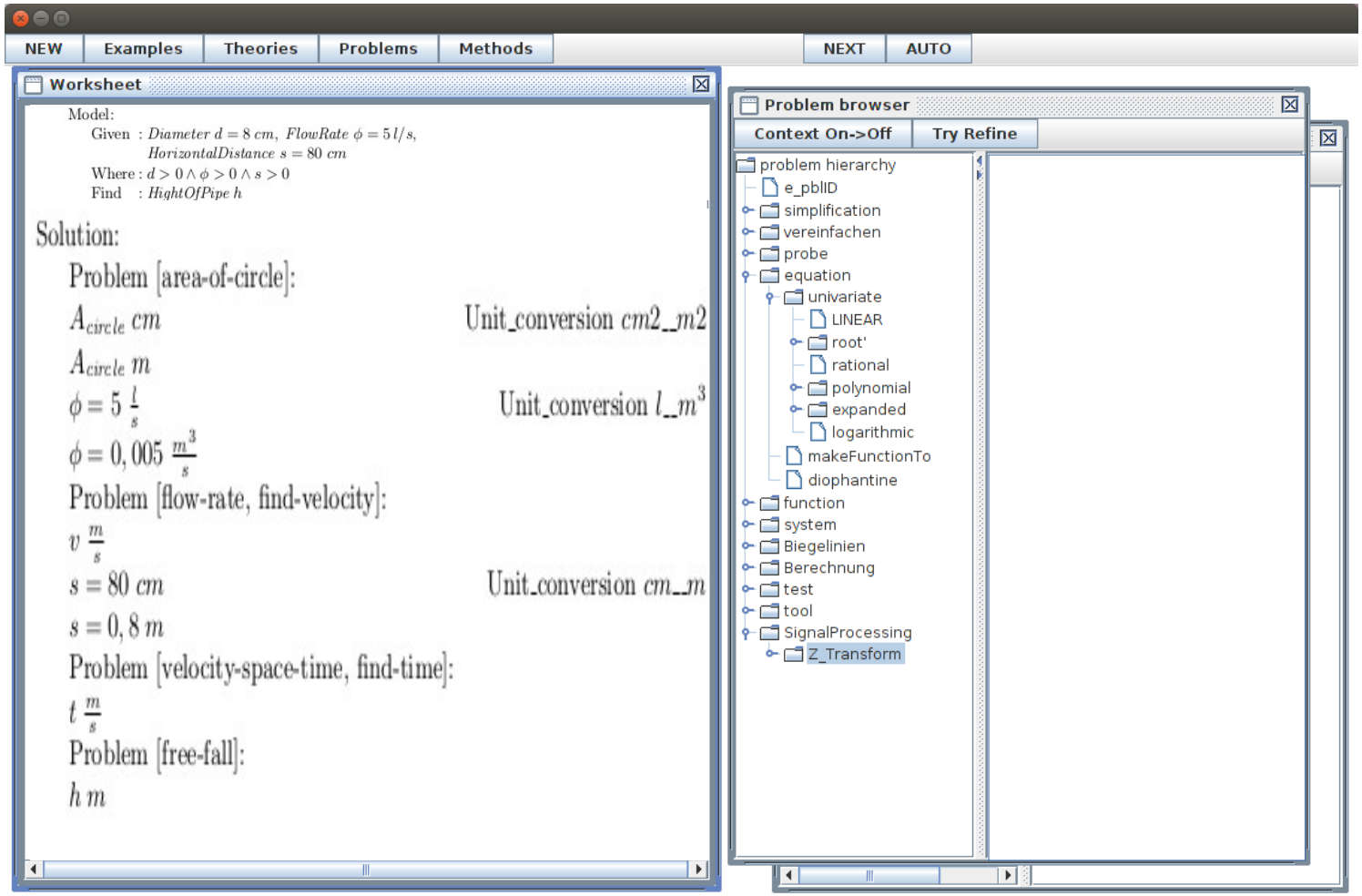
All connections finished



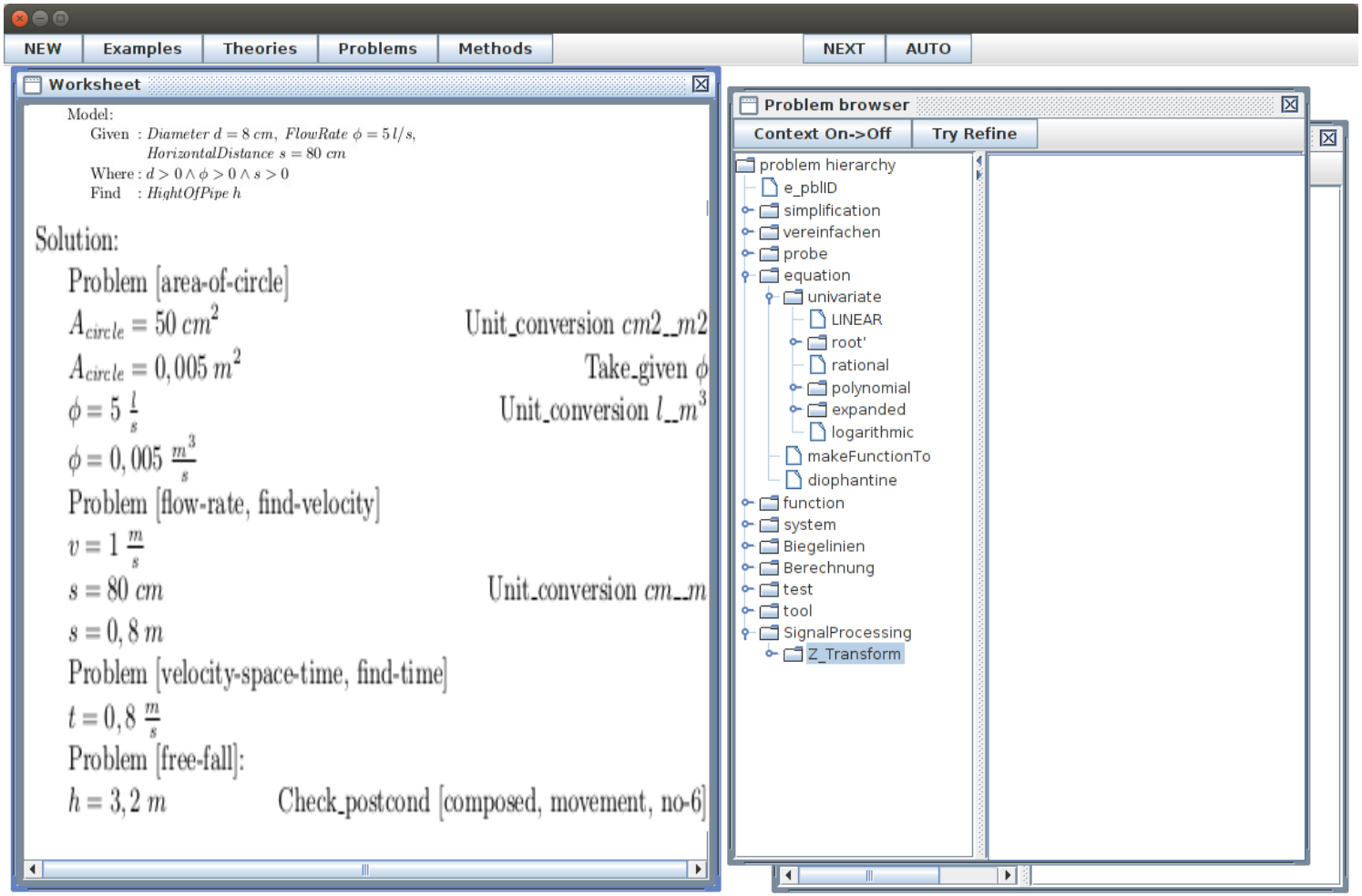
Care about unit conversions



Solution with units only



Solution complete



Outline

“Next step guidance” . . .

- . . . in specifying a problem:

  If **data for each variant** for constructing a specification
  (one variant shown above) are given,
  then the system can guide the student in completing
  a specification

- . . . in step-wise constructing a solution:

  If a **program** describes how to solve a problem
  defined by a formal specification,
  then this program run by **Lucas-Interpretation**

  - determines a next step (if requested by the student)
  - checks input of the student using the logical context.

“Next step guidance” . . .

- . . . in specifying a problem:

  If **data for each variant** for constructing a specification (one variant shown above) are given,
  then the system can guide the student in completing a specification

- . . . in step-wise constructing a solution:

  If a **program** describes how to solve a problem defined by a formal specification,
  then this program run by **Lucas-Interpretation**

  - determines a next step (if requested by the student)
  - checks input of the student using the logical context.

Outline

Programming Language . . .

. . . for authors of mathematics knowledge.

Isabelle provides a **"function package"** for programming.
Added value of this implementation:

- syntax errors are indicated accurately
- type annotations shift into the initial signature
- less type annotations are required
- syntax highlighting specific for constants etc
- free variables on right-hand-sides are rejected

Students might watch progress within a solution
like in a debugger (on request).

Conclusions

TP technology provides mechanical explanations due to

- principal benefits
- added value of implementation
- and “next step guidance”

of various kinds on different language layers — all explanations come **on users’ request**!

Field tests will show, whether “systems that explain themselves” meet the promise to make learning mathematics a game like learning to play chess by software.

In order to get prototypes ready for field tests, understanding by stakeholders in STEM education is needed, private demos of Isabelle/*ISAC* are welcome.

Conclusions

TP technology provides mechanical explanations due to

- principal benefits
- added value of implementation
- and “next step guidance”

of various kinds on different language layers — all explanations come **on users’ request**!

Field tests will show, whether “systems that explain themselves” meet the promise to make learning mathematics a game like learning to play chess by software.

In order to get prototypes ready for field tests, understanding by stakeholders in STEM education is needed, private demos of Isabelle/*ISAC* are welcome.

Conclusions

TP technology provides mechanical explanations due to

- principal benefits
- added value of implementation
- and “next step guidance”

of various kinds on different language layers — all explanations come **on users’ request**!

Field tests will show, whether “systems that explain themselves” meet the promise to make learning mathematics a game like learning to play chess by software.

In order to get prototypes ready for field tests, understanding by stakeholders in STEM education is needed, private demos of Isabelle/$\mathcal{ISAC}$ are welcome.

Thank you for Attention!