Angelica Demo
with focus on
Symbolic Propagation

Björn Hägglund
Linköping University

Top Design Goals

- Brightness (ease of use)
- Solidity (solver strength)
- Extensibility (definitional power)

Presentation Outline

- What is Angelica?
- Design goals
- Feature summary.
- Demo with focus on symbolic propagation.
- Theory of symbolic propagation.
- Questions.

Brightness

- Interactive user interface
- Appealing constraint specification language:
  - Pure.
  - Readable
  - Orthogonal

What is Angelica?

- An integrated constraint-solving environment (CSE).
- Not a programming environment.
- But final target is the domain of all executable efficient computer programs.

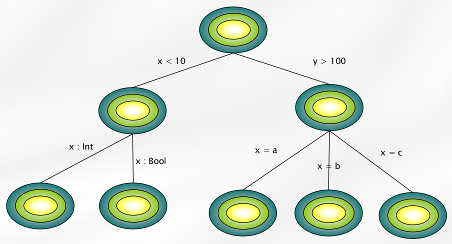
Interactivity Features

- Continuous solving with immediate screen updates during problem specification.
- Modifiable specifications.
- Colors are used to display world and propagator states.
- Entries can be cloned, rearranged and piled in any order.
- Fragmented selections.
- Type system protects against bad or trivial assertions.
- Windows can be teleported across the multiverse.

CSL Features

- Completely declarative.
- Mathematical syntax (e.g. chain-associative operators).
- Evaluation based on typed deterministic rewriting.
- Same syntax for queries and assertions.
- Everything nests and is first class.
- Variable domains are types and vice versa.
- Advanced type system:
  - Type inference.
  - Union, intersection, negation.
  - Subtyping.

Multiverse Example



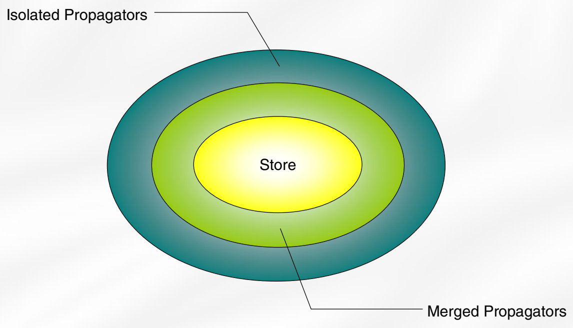
Solidity Features

- Partial evaluation.
- Parallel search with propagator-aware distribution.
- Symbolic propagation.

Store Entry Structure

- There are two kinds of store entries (stored constraints):
  - Bindings (substitutions).
  - Ascriptions (domain constraints).
- A binding equates a variable and a term. Examples:
  - x = 3
  - y = y + z
- An ascription constrains a variable to have a type denoted by some term. Examples:
  - x : Int
  - x : 2..9
  - x : 2..a -> 9..()

Universe Layout



Main Store Design Issue

- What kind of terms makes sense to allow in the rhs of a stored constraint?
- Being too restrictive severely cripples propagation.
- Being too permissive makes critical store operations ridiculously expensive or even undecidable and/or may, ironically, cripple propagation as well.

Contribution & Thesis

- By being far more permissive than what is typically the case in solvers based on search & propagation with computation spaces,
- the strength of such solvers can be increased immensely,
- without loosing any of the their essential advantages.

Store Consistency Rules

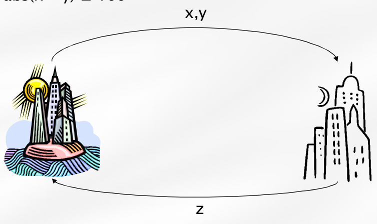
- Store must never contain cycles.
- There must never be more than one entry for the same variable.
- Whenever the store constrains a variable to a potentially empty domain, there must be at least one propagator ensuring that the domain is in fact not empty.

Example Problem

x, y, z :: 0..()

44x + 73y – 52z = 1936!
30z – 46y – 16x = 4712!

abs(x – y) ≤ 100



Cycle Avoidance Rules

- The store never binds a variable to a term containing that very variable or any other bound variable.
- Stored ascriptions must not contain variables at all.

General Store Restrictions

- The store must never become inconsistent.
- The store should never become jammed.
- Critical store operations should have a reasonable computational complexity that is independent of the store size:
  - Adding information about a variable.
  - Retreiving the information stored about a variable.
  - Triggering suspended propagators.
  - Notifying suspended propagators (firing triggers).
  - Determining the state of the store.
- The rules for designing well-behaved propagators should be understandable and achievable.

Cycle Removal Examples

x = x + y!

x = x + x y!

fac = λx:Int. if x = 0 then 1 else x fac(x – 1)!

even = λx:Nat. if x = 0 then true else odd(x – 1)!
odd = λx:Nat. if x = 0 then false else even(x – 1)!

x = 1, 2 » x!

x = (1) » x » (2)!

Ruling Out Multiple Entries

- New bindings of an already bound variable are always rejected. Instead, propagators are expected to use the old binding to obtain a new equation.
- When binding a constrained variable, the new binding is always accepted but the already stored ascription is expelled from the store. The propagator that added the new binding is expected to take care of the expelled ascription.
- If the store receives an ascription of an already constrained variable, the two ascription entries are merged using type intersection. The sending propagator gets a compatibility test in return.

Some Accepted Terms

2	
Int -> Int	
λx.x	
x	
1, 2, a, 5, 6, b, 8, 9	
-x	
x + y	
2x	
6x+ 4y + 7z	(if x,y,z :: Int)
1, 2 » a » 8, 9	
1, 2 » a » 5, 6 » b » 8, 9	(if length of a or b is known)

Rejecting Jamming Terms (1)

- A variable may only be bound to accepted terms.
- A term T is *accepted* only if it is final or it is true for all accepted terms S that unifying S with T is guaranteed to fail or add information to the store enabling the reduction of S and T to equal terms.

Some Rejected Terms

x y
f x
if a then b else c

Rejecting Jamming Terms (2)

- A term T is *accepted* only if all terms more precise than T are accepted.
- A term S is more precise than a term T iff the range of S is a proper subset of the range of T.
- The *range of* a term is the set of all terms to which the term may be eventually reduced.

Note on Store Updates

- To avoid expensive recomputations, only irreducible (suspended or final) terminating terms are allowed in the rhs of a store entry.
- Since suspended terms may later become reducible, store entries need to be updated.
- Store updates are triggered, launched, executed and scheduled like propagators. Hence, the immediate complexity of the critical store operations remains independent of the store size.