Finite Domain Constraints in SICStus Prolog

Mats Carlsson

Swedish Institute of Computer Science

matsc@sics.se

http://www.sics.se/~matsc

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

CICLOPS 2001

Outline of the Talk

- The SICStus `library(clpfd)` Package
  - built-in primitives
  - implementation architecture
  - indexicals
  - global constraints
- Host Language Support
- Internal Representation
  - domain variables
  - propagation queues
- Stateful Constraints
  - unification and co-references
  - optimizations
- Debugging
- Conclusion

CLP over Finite Domains

- Constraint store
  - $X \subseteq D$, $D \subseteq \mathbf{Z}$
- Terms
  - integers (can be <0)
  - variables ranging over finite domains
- Constraints
  - linear arithmetic constraints
  - combinatorial constraints
  - reified constraints: $p(x_1,\ldots,x_n) \Leftrightarrow b$
  - propositional combinations of reified constraints
  - user-defined constraints

Built-in Constraints

```
element/3
case/[3,4]
all_different/[1,2]
assignment/[2,3]
circuit/[1,2]
cumulative/[4,5]
serialized/[2,3]
disjoint1/[1,2]
disjoint2/[1,2]
cumulatives/[2,3]
global_cardinality/2
count/4
scalar_product/4
sum/3
knapsack/3
```

```
X in Domain, X in_set Set

X #= Y, X #\= Y
X #< Y, X #=< Y
X #>= Y, X #> Y
```

```
#\ C
C #/\ D
C #\/ D
C #=> D
C #<=> D
```

```
C #<=> B
```

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

Built-in Search

- `indomain(Var)`
- `labeling(Options,Vars)`
- `minimize(Goal,Var)`
- `maximize(Goal,Var)`

Labeling options

- `leftmost | min | max | ff | ffc | variable(Sel)`
- `enum | step | bisect | up | down | value(Enum)`
- `discrepancy(D)`
- `all | minimize(Var) | maximize(Var)`

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

Implementation Architecture

- A scheduler for **indexicals** and **global constraints**
- Support for **reified constraints**
- User-defined indexicals for fine-tuned propagation within a general framework
- Global constraints use specialized filtering algorithms
- Custom designed suspension mechanism
- Support for stateful constraints

Indexicals

- Given a constraint $C(X_1,\ldots,X_n)$, for each X_i, write a rule X_j *in* R_j that computes the feasible values of X_i in terms of $\{dom(X_i) \mid i \neq j\}$.
  - [VSD92] P. Van Hentenryck, V. Saraswat, Y. Deville. Constraint processing in cc(FD), 1992. Draft.
- Example: $X = Y + C$, domain consistent version.

```
eqcd(X,Y,C) +:
   X in dom(Y)+C,
   Y in dom(X)-C.
```

- Example: $X = Y + C$, interval consistent version.

```
eqcd(X,Y,C) +:
   X in min(Y)+C..max(Y)+C,
   Y in min(X)+C..max(X)-C.
```

Indexicals: Pros and Cons

- Feasibility demonstrated by D. Diaz: clp(FD), GNU Prolog
- Other implementations by G. Sidebottom, H. Lock, H. Vandecasteele, B. Carlson, ...
- A RISC approach to constraint solving
- Reactive functional rules executed by a specialized virtual machine
- A language for fine-tuned propagation in a general framework
- A language for **entailment detection** and hence **reification**
- Drawbacks:
  - low granularity
  - local effect
  - fixed arity

Indexicals: Definitions

- R_S denotes the range expression *R* evaluated in the constraint store *S*
- *S'* is an **extension of *S*** iff

$$\forall X : dom(X)_{S'} \subseteq dom(X)_S$$

- *R* is **monotone in *S*** iff for every extension *S'* of *S*,

$$R_{S'} \subseteq R_S$$

- *R* is **anti-monotone in** *S* iff for every extension *S'* of *S*,

$$R_S \subseteq R_{S'}$$

Indexicals: Syntax of `X in R`

Range expressions

```
R ::= T..T | R/\R | R\/R | \R | R+T | R-T | R mod T | {T,…,T}
    | dom(X)
```

Term expressions

```
T ::= T+T | T-T | T*T | T/>T | T</T | T mod T | min(X) |
    max(X) | card(X) | X | N
```

```
N ::= integer | inf | sup
```

Monotonicity

Indexicals for **constraint solving** must be **monotone**

Indexicals for **entailment detection** must be **anti-monotone**

Indexicals for Reification

- Example: X = Y + C.

```
?- eqcd(X,Y,5) <=> B.

eqcd(X,Y,C) +:       % positive constraint solving
   X in dom(Y)+C,
   Y in dom(X)-C.
eqcd(X,Y,C) -:       % negative constraint solving
   X in \{Y+C},
   Y in \{X-C}.
eqcd(X,Y,C) +?       % entailment detection
   X in {Y+C}.
eqcd(X,Y,C) -?       % disentailment detection
   X in \dom(Y)+C.
```

Indexicals: Implementation

- Compiled to (bytecode,symbol table).
- Indexical syntax intercepted by `user:term_expansion/2`

```
user:term_expansion((Head+:Body), Expansion) :-
   functor(Head, N, A),
   Expansion = [:- clpfd:'$fd_install'(N/A, 1, Info)],
   compile(Head, Body, Info).
```

- Executed by a simple stack-based VM.
- `eqcd/3` gets defined as a **Prolog predicate**
  - the WAM escapes to a solver entrypoint

The Global Constraints API

- `fd_global(+C,+S,+V)`
  - Posts a global constraint C with initial state S; V tells how to suspend on variables by means of a list of

    `dom(X), min(X), max(X), minmax(X), val(X)`
- `clpfd:dispatch_global(+C,+S0,-S,-A)` **User defined**.
  - Entrypoint for the filtering algorithm of global constraint C with state S0, producing a new state S and solver requests A (entailed, disentailed, prune, …).
- `fd_min(?X,-Min), fd_max(?X,-Max), ...`
  - Unifies Min (Max) with the current lower (upper) bound of X.
- FD set ADT
  - Comes with all the necessary operations.

$x \leq y \Leftrightarrow b$ as a Global Constraint

```
le_iff(X,Y,B) :-
   B in 0..1,
   fd_global(le(X,Y,B), [], [minmax(X),minmax(Y),val(B)]).

:- multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(le(X,Y,B), [], [], Actions) :-
    (     var(B)
    ->    (    fd_max(X,Xmax), fd_min(Y,Ymin), Xmax =< Ymin
          ->   Actions = [exit,B=1]             % entailed, B=1
    ->    (    fd_max(Y,Ymax), fd_min(X,Xmin), Xmin > Ymax
          ->   Actions = [exit,B=0]             % entailed, B=0
          ;    Actions = []                     % not entailed, no pruning
          )
    ;     B=:=0
    ->    Actions = [exit,call(X#>Y)]           % rewrite to X#>Y
    ;     Actions = [exit,call(X#=<Y)]          % rewrite to X#=<Y
    ).
```

Outline of the Talk

- The SICStus `library(clpfd)` Package
  - built-in primitives
  - implementation architecture
  - indexicals
  - global constraints
- Host Language Support
- Internal Representation
  - domain variables
  - propagation queues
- Stateful Constraints
  - unification and co-references
  - optimizations
- Debugging
- Conclusion

Generic Support

- **Backtracking, trailing**
  - Provides *search*, automatic *memory reclamation, state restoration, do-on-backtracking*
- **Meta-calls, encapsulated computations**
  - Enables meta-constraints
    - *cardinality-path* [Beldiceanu&Carlsson, ICLP2001]
    - *Satisfiability Sum* [Régin et al., CP2001]
- **Term Expansion**: `user:term_expansion/2`
  - Recognizes and translates indexical “clauses”
- **Goal Expansion**: `user:goal_expansion/3`
  - Provides macro-expansion
  - Recognizes and translates arithmetic constraints
    - `X #= Y, X #>= Y`, etc.
  - Recognizes and translates propositional constraints
    - `P #/\ Q, P #\/ Q`, etc.

Support Targeted for CLP

- **Attributed Variables** provide the link from unification to solvers, and allow solvers to store data on variables.
  - C. Holzbaur. *Specification of Constraint Based Inference Mechanism through Extended Unification*. PhD thesis, U. of Vienna, 1990.
  - Unification hooks
  - Top-level loop hooks

```
:- attribute fd_attribute(_,_).

?- get_atts(X, fd_attribute(DomMut,SuspMut)).
?- put_atts(X, fd_attribute(DomMut,SuspMut)).

verify_attributes(Var, Term, Goals) :- ...
```

Support Targeted for CLP

- **Mutable Terms** provide backtrackable assignment (value-trailing).
  - N. Beldiceanu, A. Aggoun. *Time Stamps Techniques for the Trailed Data in CLP Systems.* Actes du Séminaire 1990 - Programmation en Logique, Tregastel, France.
  - Only for Prolog terms, not arbitrary memory locations
  - Coarse trailing [Choi, Henz and Ng, CP2001]

```
'$mutable'(Term,Timestamp)

create_mutable(+Term,+Mutable)
get_mutable(+Term,+Mutable)
update_mutable(+Term,+Mutable)
```

Outline of the Talk

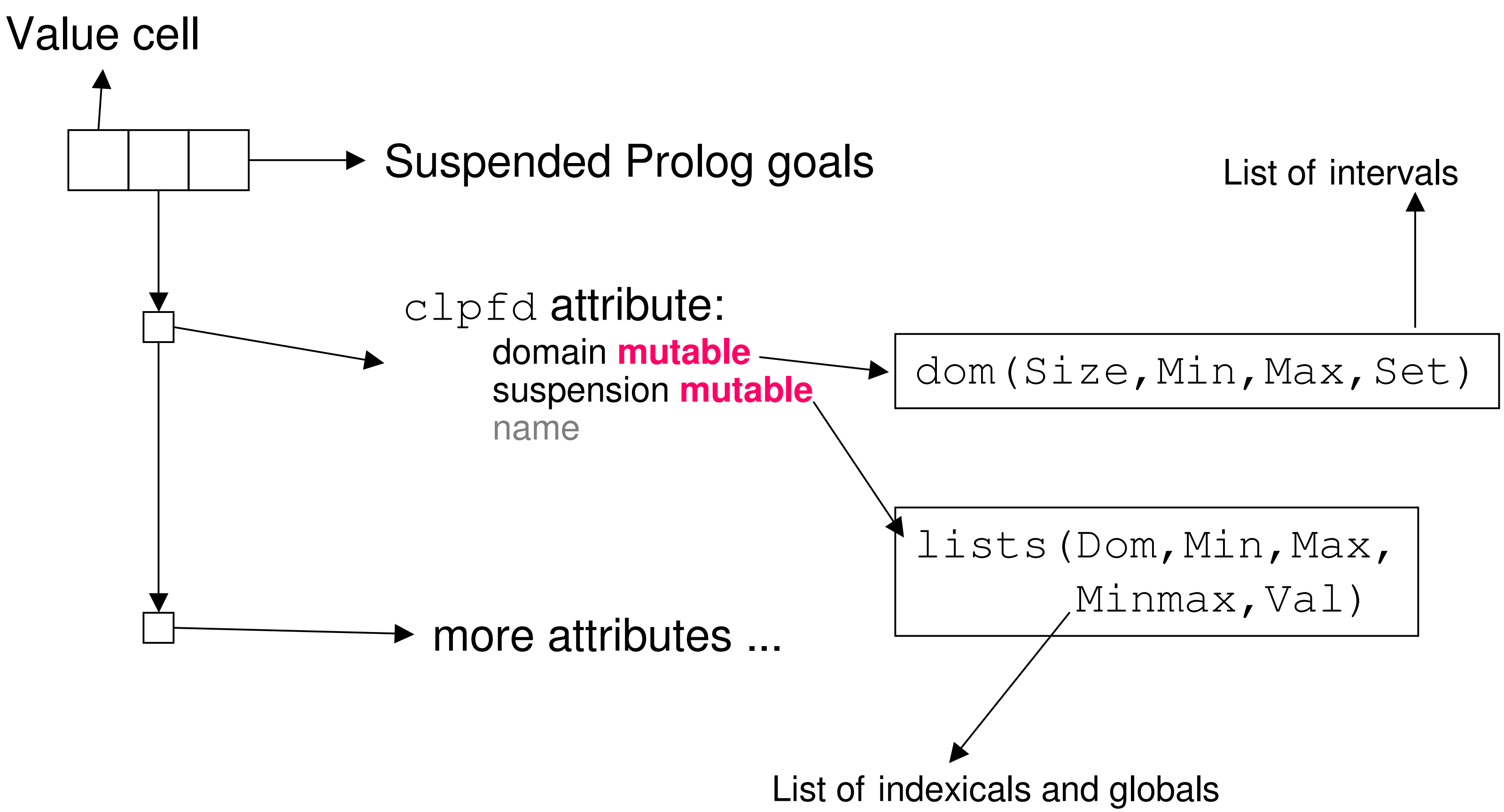
- The SICStus `library(clpfd)` Package
  - built-in primitives
  - implementation architecture
  - indexicals
  - global constraints
- Host Language Support
- Internal Representation
  - domain variables
  - propagation queues
- Stateful Constraints
  - unification and co-references
  - optimizations
- Debugging
- Conclusion

Domain representation

- Options:
  - interval+bit array [CHIP compiler, clp(FD), GNU Prolog, CHOCO, Mozart]
  - array of integers [CHIP compiler]
  - **list of intervals** [ECLiPSe, SICStus,CHOCO,Mozart,MROPE,Figaro]
  - interval trees [CHOCO]
  - interval only [interval solvers, CHIP compiler]
  - interval + list of holes [?]

- Pros (assuming *M* intervals)
  - operations *O(M)* in the worst case
  - implementation straightforward
  - Prolog representation straightforward
  - scalable
- Cons
  - performs poorly on *N Queens*

Domain Variables



Propagation Queues

- Queues of **constraints**, not **variables**
  - The KISS principle
  - One indexical queue (greater priority)
  - One global constraint queue (lesser priority)
- Enqueued test in O(1) time
  - using a **mutable term**
- **No extra information** stored with queue elements
  - which variables were pruned
  - why they were pruned
  - their previous domains

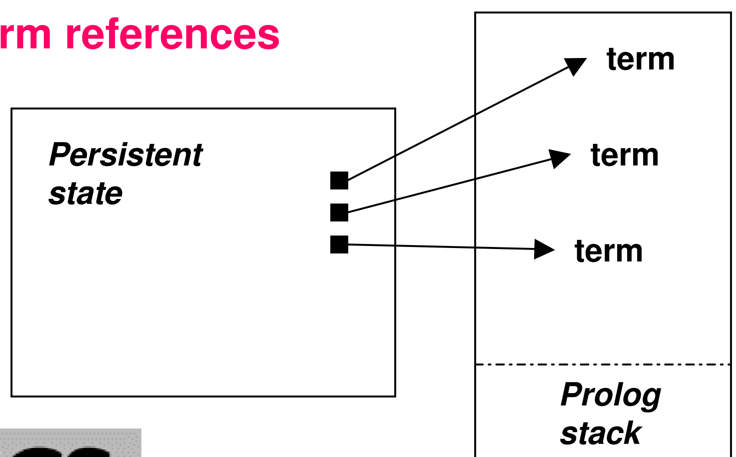
- Historically, **difference lists** were being passed around
- Now using **dedicated buffers**
  - modest performance gains
  - needs garbage collector services

Outline of the Talk

- The SICStus `library(clpfd)` Package
  - built-in primitives
  - implementation architecture
  - indexicals
  - global constraints
- Host Language Support
- Internal Representation
  - domain variables
  - propagation queues
- → Stateful Constraints
  - unification and co-references
  - optimizations
- Debugging
- Conclusion

Stateful Constraints

- `clpfd:dispatch_global(+Ctr,+S0,-S,-A)` **User defined**.
  - Entrypoint for the filtering algorithm of global constraint Ctr with state S0, producing a new state S and solver requests A.
  - Does not say **which** domain variables were pruned.
  - Provides for state as a **Prolog term**. However, most built-in constraints are written in C $\Longrightarrow$ costly conversion to C data **each time** Ctr wakes up.
- Persistent state in C, requiring:
  - **deallocation guaranteed** on backtracking or determinate entailment
  - global **term references**



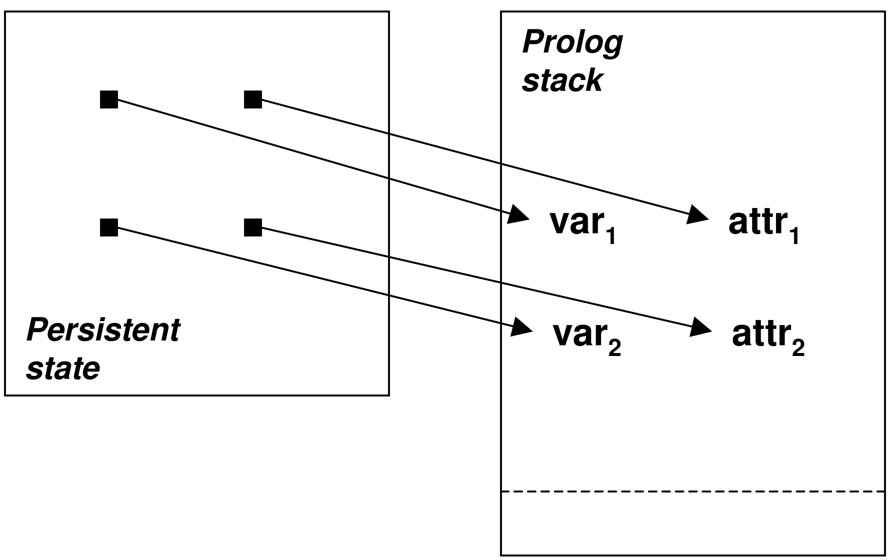
Support for Stateful Constraints

- Global **term references**
  - explicitly allocated and deallocated
  - requires garbage collector support
  - dangling pointer hazard if used generally

- **Deallocation guaranteed**
  - on backtracking
  - on determinate entailment
  - both the memory block and the global term references

Domain Variables in the Persistent State

- For each domain variable, we store
  - one term reference to the variable itself
  - one term reference to the attribute term
- Why?
  - Look up attribute term once only
  - Retain access to attribute even if the variable is ground



Pruning in Global Constraints

```
clpfd:dispatch_global(+C,+S0,-S,-A)
```

where `A` is a list of:

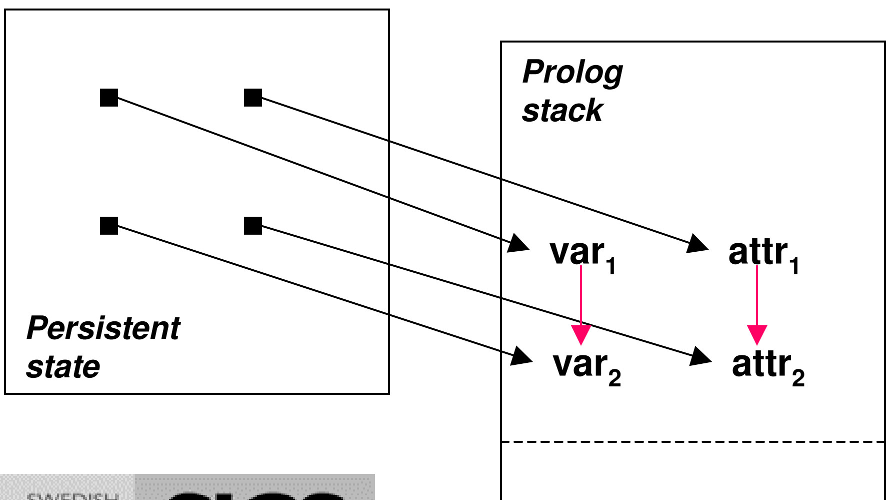
```
X in Domain, X in_set Set, X=Int, call(Goal), exit,fail
```

- Direct pruning *inside* filtering algorithm is not allowed.
- Three-phase pruning scheme:

1. At entry, make local “copies” of the domain variables.
2. The algorithm works with the local “copies”.
3. At exit, results are posted by computing `A`.

Handling Unification and Co-References

- Variable-variable unifications require:
  - forwarding one attribute to another
  - forming intersection of domains
  - forming union of suspensions
  - waking up relevant constraints
  - marking relevant constraints as having co-references
  - in C: dereferencing attributes as well as variables



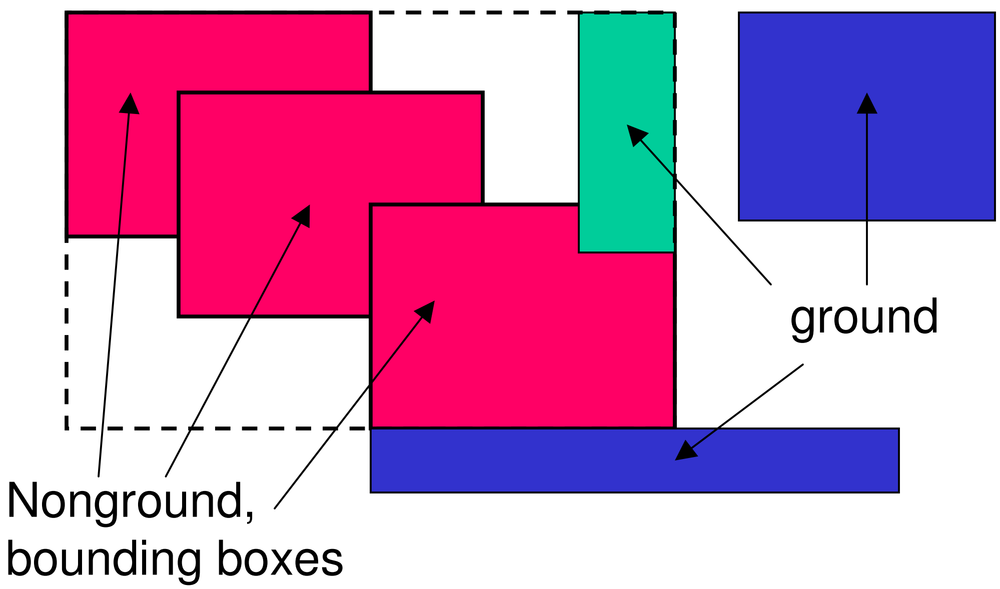
Filtering Algorithms and Co-References

- Each filtering algorithms is assumed to reach a fixpoint if no domain variable occurs more than once.
  - The constraint normally does not wake itself up.

- If there are co-references, the solver will repeat the filtering algorithm until no more pruning.
  - The constraint wakes itself up.
  - domain variables occurring more than once initially
  - co-references introduced by unification

Generic Optimization: Sources & Targets

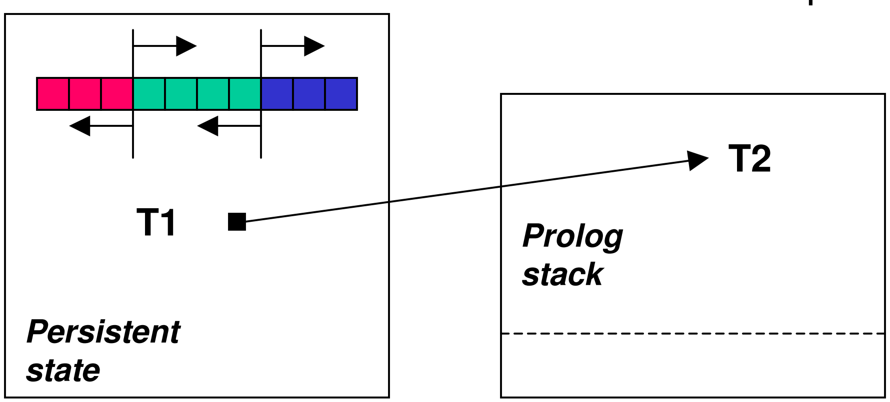
- A **target** object is subject to pruning or check
- A **source** object can lead to some pruning or check
- **Inactive** objects can be ignored
- Speedup > 2.5 observed for non-overlapping rectangles



SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

Generic Optimization: Incrementality

- If the current store in an extension of the previous one, then
  - ground/source/inactive objects stay so
- Otherwise,
  - recompute (part of) the persistent state
- If no choicepoints younger than the posting time of the constraint
  - ground/source/inactive objects stay so **forever**
- Detecting the incremental case:
  - timestamps: T1 in C, T2 in a **mutable term**, T1 := T2 := T2+1 at exit
  - the current store is an extension of the previous one if T1=T2 at entry



Outline of the Talk

- The SICStus `library(clpfd)` Package
  - built-in primitives
  - implementation architecture
  - indexicals
  - global constraints
- Host Language Support
- Internal Representation
  - domain variables
  - propagation queues
- Stateful Constraints
  - unification and co-references
  - optimizations
- → Debugging
- Conclusion

A Finite Domain Constraint Tracer

- Provides:
  - tracing of selected constraints
  - naming of domain variables
  - Prolog debugger extensions (naming variables, displaying annotated goals)

- Default appearance (customizable):

```
scalar_product([0,1,2,3],[1,<list_2>,<list_3>,<list_4>],#=,4)
   list_2 = 1..3
   list_3 = 0..2 -> 0..1
   list_4 = 0..1
```

- Comes with SICStus Prolog 3.9

Towards Better Debugging Tools

- Starting point: fine-grained execution trace
  - the DiSCiPl experience
- Drawbacks:
  - rough explanations (unary constraints)
  - flat sequence of low-level events
  - static information missing
    - the constraints themselves
    - the way constraints are woken
    - what kind of pruning constraints do
    - what kind of consistency the constraints achieve
    - what type of filtering algorithms they use
  - no means of considering subparts of global constraints to improve explanations
    - specific necessary conditions
    - specific methods used
    - implied constraints

Prerequisites for Better Debuggers

- Static information about constraints
  - the way they are woken
  - what kind of pruning they do
  - what kind of consistency they achieve
  - details about the filtering algorithms they use
- Status information
  - status of constraints
    - e.g. suspended, entailed, failed
  - status of variables
    - e.g. infinite domain, finite domain, interval, ground
- Trace information
  - the events that occur during execution
  - **explanations** for these events
  - structured

Towards Better Explanations

- Challenges:
  – record **multiple explanations** for each value removal compactly
  – give explanations in terms of **non-unary constraints**
  – give explanations in terms of **objects of the applications**

"To fix this failure, you should modify the origin *attribute of at least 3 tasks out of this set of 5 tasks."*

- Uses:
  – non-chronological backtracking
  – focused explanations to the user
  – propose which constraints to relax to fix a failure
  – propose which constraints to relax or enforce in over-constrained problems

Conclusion: what's crucial for a good CLP(FD) system

- Generic host language support
  - attributed variables, mutables, term and goal expansion
- A good foreign language interface
- Support for persistent foreign language state
  - do-on-backtracking, persistent term references
- Good debugging facilities
- Nicolas Beldiceanu

- The full story at:

http://www.sics.se/sicstus