Constraint-Based Program Analysis

Alex Aiken
University of California, Berkeley

Nevin Heintze
Carnegie Mellon University

URL http://http.cs.berkeley.edu/~aiken/popl95.ps.Z

What is This Talk About?

- Part technical overview.
  - Survey of results in constraint-based program analysis.
  - Discussion of algorithms, directions, open problems.
- Part systematization.
  - Great diversity in constraint-based program analyses.
  - Many results share common foundations.
- Part shameless advertisement ...
- ... for set constraints, the speakers' pet theory.

What is Program Analysis?

- For this talk, we take a broad definition.

- A program analysis =
  any syntax-directed analysis.

- Semantic framework: Abstract interpretation
  (Cousot & Cousot)

- Paradigms for constructing analyses:
  - Data-flow analysis.
  - Type inference.
  - Fixed-point algorithms.
  - ∗ Constraint-based analysis.

What is Constraint-Based Program Analysis?

- A constraint-based analysis:
  - Generates local constraints from the program text.
  - Solutions of the constraints yield the desired information.

- Separates specification and implementation.

- Specification = Constraint generation.
  - Local constraints $\Rightarrow$ tractable, structural proofs.
  - Hides details of implementation.

But There's More ...

- Implementation = Constraint resolution.
  - The place for algorithms and efficiency concerns.
  - A pure constraint problem, independent of high-level application.
- Constraint resolution not tied to program text.
- Bottom line: Constraint point of view useful for specification and implementation.

Example: Data-Flow Analysis

- Consider live-variable analysis for C programs.
- Domain: Set of program variables.
- Variables: $[\![S]\!]_{in}$, $[\![S]\!]_{out}$
- Constants: S_{def}, S_{use}
- Constraints for each program statement S:

$$
\begin{aligned}
[\![S]\!]_{in} &= S_{use} \cup ([\![S]\!]_{out} - S_{def}) \\
[\![S]\!]_{out} &= \bigcup_{X \in succ(S)} [\![X]\!]_{in}
\end{aligned}
$$

- Easy to show constraints have a least solution.

Example: Type Inference

- Consider Hindley/Milner type inference for lambda calculus.
- Domain of finite types: $\tau ::= \alpha \mid \tau_1 \rightarrow \tau_2$
- A variable $[\![e]\!]$ for each expression e.
- Equations of the form

$$\begin{aligned} [\![\lambda x.e]\!] &= [\![x]\!] \rightarrow [\![e]\!] \\ [\![e_1]\!] &= [\![e_2]\!] \rightarrow [\![e_1\ e_2]\!] \end{aligned}$$

- Easy to solve equations using unification.

Two Constraint Theories

- Data-flow Analysis
  - Domain is sets of atoms.
  - Constraints over expressions of variables and set operations.
- H/M Type Inference
  - Domain is sets of terms.
  - Equations over terms.
- Two important basic theories, developed and studied independently over many years.
- But are they really independent?

Set Constraints

- Set of constructors $C = \{b, c, \ldots\}$
  Set of variables $V = \{\alpha, \beta, \ldots\}$

- The *set expressions* are

$$
\begin{aligned}
E \ ::= \ & 0 \,|\, \alpha \,|\, E_1 \cup E_2 \,|\, E_1 \cap E_2 \,|\, \neg E_1 \\
& |\, c(E_1, \ldots, E_n) \,|\, c_i^{-1}(E_1)
\end{aligned}
$$

- A system S of set constraints is

$$\bigwedge_i E_i \subseteq E_i'$$

- Question: What are the solutions of S?

Semantics of Set Expressions

$$E ::= 0 \,|\, \alpha \,|\, E_1 \cup E_2 \,|\, E_1 \cap E_2 \,|\, \neg E_1$$
$$|\, c(E_1, \ldots, E_n) \,|\, c_i^{-1}(E_1)$$

- One interpretation: Each set expression denotes a subset of H, the Herbrand universe of terms.
- An *assignment* σ is a function $V \rightarrow 2^H$.
- Extend σ to set expressions:

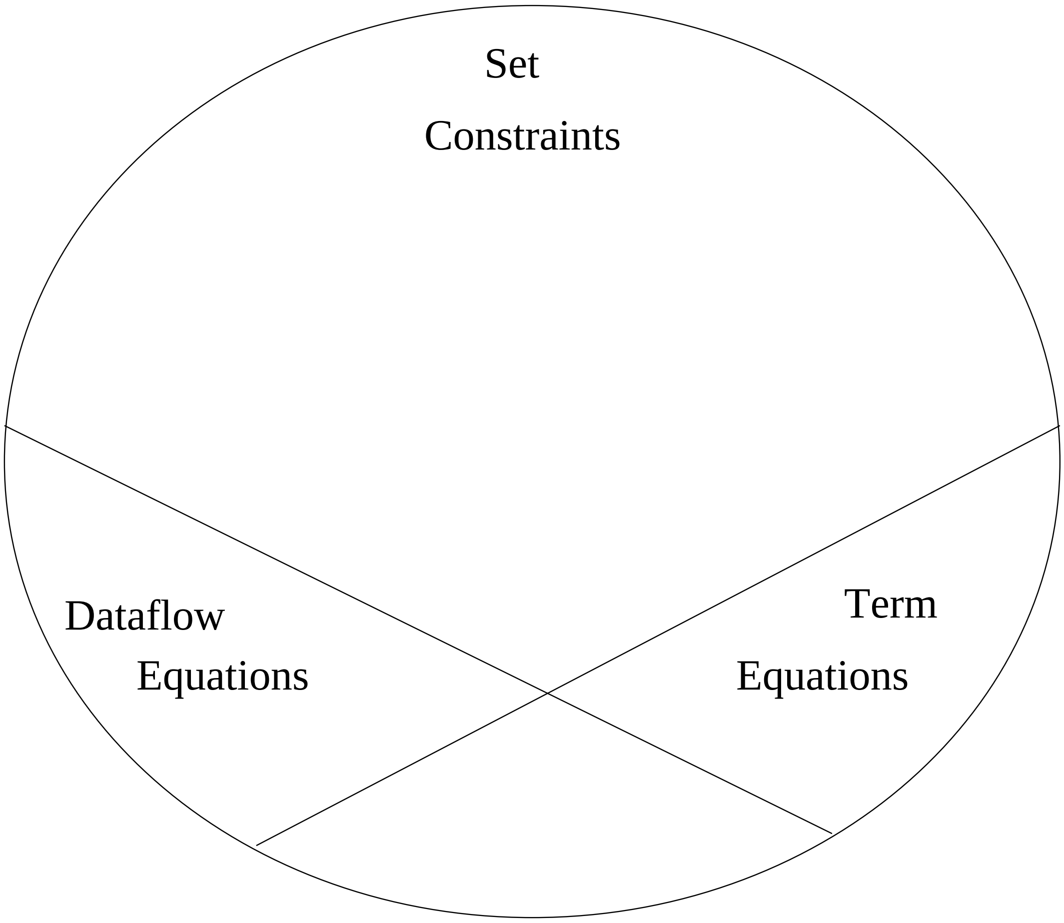
$$\begin{aligned}
\sigma(0) &= \emptyset \\
\sigma(E_1 \cup E_2) &= \sigma(E_1) \cup \sigma(E_2) \\
\sigma(E_1 \cap E_2) &= \sigma(E_1) \cap \sigma(E_2) \\
\sigma(\neg E) &= H - \sigma(E) \\
\sigma(c(E_1, \ldots, E_n)) &= \{c(t_1, \ldots, t_n) | t_i \in \sigma(E_i)\} \\
\sigma(c_i^{-1}(E)) &= \{t_i | c(t_1, \ldots, t_n) \in \sigma(E)\}
\end{aligned}$$

- σ is a *solution* of constraints $\bigwedge_i E_i \subseteq E'_i$ iff

$$\forall i \; \sigma(E_i) \subseteq \sigma(E'_i)$$

Three Constraint Theories

- Set Constraints are a generalization of
- ... data-flow equations (adds data structures)
- ... term unification (adds set operations)
- ... and more (projections)



A Thesis

- Set constraints are a natural formalism for many constraint problems in program analysis.
- Useful because it includes many interesting sub-theories.
- Results encapsulate much of what is "hard" about solving constraints
  $\Rightarrow$ off-the-shelf results for analysis algorithms.
- Puts diverse analyses in a common framework.
- And leads directly to new analyses ...

Something New

- Lambda calculus again.
- Variable $[\![e]\!]$ for each expression e
- Types: $\tau = \alpha \,|\, \tau_1 \rightarrow \tau_2 \,|\, \tau_1 \cap \tau_2 \,|\, \tau_1 \cup \tau_2 \,|\, 0$
- Constraints:

$$[\![\lambda \mathsf{x}.\mathsf{e}]\!] \supseteq [\![\mathsf{x}]\!] \rightarrow [\![\mathsf{e}]\!]$$
$$[\![\mathsf{e}_1]\!] \subseteq [\![\mathsf{e}_2]\!] \rightarrow [\![\mathsf{e}_1\ \mathsf{e}_2]\!]$$

- Examples:
  $\vdash \lambda \mathsf{f}.\lambda \mathsf{x}.\mathsf{f}(\mathsf{f}(\mathsf{x})) : ((\alpha \rightarrow \beta) \cap (\beta \rightarrow \gamma)) \rightarrow \alpha \rightarrow \gamma$
  $\vdash \mathsf{Y}\ \mathsf{id} : 0$
- Notes
  - Generalizes H/M type inference.
  - Straightforward to add polymorphism.
  - Constraint resolution in $\mathcal{O}(n^3)$ time.
  - (Herbrand model is not adequate, though.)

What About Projections?

- No examples with projections yet.

- Projections model selectors of data structures. (e.g., "hd" and "tl")

- But projections have another interesting property:

$$\mathsf{c}_1^{-1}(\mathsf{c}(\mathsf{A},\mathsf{B})) = \begin{cases} \mathsf{A} \text{ if } \mathsf{B} \neq 0 \\ 0 \text{ if } \mathsf{B} = 0 \end{cases}$$

- Projections can encode logical implications of constraints:

$$\mathsf{B} \neq 0 \Rightarrow \mathsf{A} \subseteq \mathsf{C} \;\equiv\; \mathsf{c}_1^{-1}(\mathsf{c}(\mathsf{A},\mathsf{B})) \subseteq \mathsf{C}$$

- Such *conditional* constraints are very useful ...

Another Example: Analysis of OO Programs

- Consider class analysis of object-oriented programs.
- Domain: Sets of class names.
- A variable $[\![e]\!]$ for each expression e.
- Analysis assigns possible classes of e to $[\![e]\!]$.
- Sample constraints:

$[\![\text{new C}]\!] = \{C\}$

$[\![\text{if } e_1 \text{ then } e_2 \text{ else } e_3]\!] = [\![e_2]\!] \cup [\![e_3]\!]$

$[\![\text{id} \leftarrow e]\!] = [\![e]\!] \subseteq [\![\text{id}]\!]$

for each method application $e_0.f(e_1, \ldots, e_n)$ and
for each class C with a method $f(x_1, \ldots, x_n) = e$

$$C \in [\![e_0]\!] \Rightarrow \bigwedge_i ([\![e_i]\!] \subseteq [\![x_i]\!])$$

Note: $C \in X \equiv X \cap C \neq 0$

OO Analysis (Cont.)

- Constraint resolution in $\mathcal{O}(n^3)$ time.
- This analysis was discovered by Palsberg and Schwartzbach.
- Closely related to closure analysis for functional programs (Jones,Shivers).
- Relevant fragment of set constraints known since 1969 (Reynolds).

- Constraint-based program analysis is a Tower of Babel.
- What's missing is a common lexicon: accepted terminology and general, reusable results.

General Results on Set Constraints

- Thm Deciding whether an arbitrary system of set constraints has a solution is NEXPTIME-complete.
- There are interesting polynomial-time fragments. Consider a generalization of the examples:

$$
\begin{aligned}
L &::= L_1 \cup L_2 \mid c(L_1, \ldots, L_n) \mid \alpha \mid 0 \\
R &::= R_1 \cap R_2 \mid c(R_1, \ldots, R_n) \mid \alpha \mid 0
\end{aligned}
$$

- Let C be a conjunction of constraints of the form:

$$
\begin{aligned}
& L \subseteq R \\
\text{or} \quad & L_0 \neq 0 \Rightarrow L_1 \subseteq R_1
\end{aligned}
$$

- Thm C has a least solution computable in $\mathcal{O}(n^3)$ time.

More Results

- Where does complexity come from?
- For PTIME algorithms, transitive closure of a graph.

$$\mathsf{A} \subseteq \alpha \subseteq \mathsf{B} \Rightarrow \mathsf{A} \subseteq \mathsf{B}$$

- For EXPTIME algorithms, general $\cap, \cup, \neg$.
- For NEXPTIME algorithms, the choice:

$$\mathsf{c}(\mathsf{A}, \mathsf{B}) = 0 \Leftrightarrow \mathsf{A} = 0 \text{ or } \mathsf{B} = 0$$

- Many complexity results for various restrictions, combinations of operators.

- References: Reynolds, Jones & Muchnick, Mishra & Reddy, Heintze & Jaffar, Aiken & Kozen & Vardi &Wimmers, Bachmair & Ganzinger & Waldmann, Charatonik & Pacholski

And More Results

- Set constraints are closely related to:
- ...graphs and graph algorithms.
- ...tree automata and tree grammars.
- ...fragments of logic.
- The constraints/automata/graphs connections come up often.
- References: Gilleron & Tomasi, Kozen & Palsberg & Schwartzbach

The Case for (Set) Constraints

- A general body of useful "off-the-shelf" results.
- Demonstrably useful for
  - explaining and unifying known analyses.
  - providing insight for new analyses.
- So are set constraints the common lexicon?
- No ... but they are part of it.

Solving Set Constraints

- What do we mean by "solve"?
  - check satisfiability
  - find representation of least solution
  - find representation of all solutions

- Focus here on a particular example:
  least solutions of restricted set constraints.

Finding Least Solutions

Algorithm based on rewrite/completion rules.

- $\alpha \supseteq E \cup E' \implies \alpha \supseteq E$ and $\alpha \supseteq E'$

- $\alpha \supseteq \beta \ \wedge \ \beta \supseteq E \implies \alpha \supseteq E$

- $\alpha \supseteq \mathsf{cons}_1^{-1}(\beta) \ \wedge \ \beta \supseteq \mathsf{cons}(\gamma, \delta) \implies \alpha \supseteq \gamma$

  WRONG: this may not preserve the least model!
  e.g. suppose that $\gamma = a$ and $\delta = 0$.

Overview of Least Model Algorithm

- Goal: initial constraints $\leadsto$ solved form
- Representations of solved form
  - cyclic graphs
  - tree grammars
  - (restricted) set constraints
- Issues
  - rewrite rules: correctness, "completeness"
  - termination
- Constraints are maintained in a "normal form" during the algorithm. e.g. $\alpha \supseteq E$ where

  $$E ::= \alpha \mid c(\alpha_1, \ldots, \alpha_n) \mid c_i^{-1}(\alpha) \mid \alpha_1 \cap \alpha_2$$

Constraints and Tree Grammars

Initial:

$$
\begin{array}{rcl}
\alpha & \supseteq & \mathsf{cons}_1^{-1}(\beta) \\
\beta & \supseteq & \mathsf{cons}(\gamma, \beta) \\
\beta & \supseteq & \mathsf{nil} \\
\gamma & \supseteq & \mathsf{a} \\
\gamma & \supseteq & \mathsf{b}
\end{array}
$$

Added:

$$
\begin{array}{rcl}
\alpha & \supseteq & \gamma \\
\alpha & \supseteq & \mathsf{a} \\
\alpha & \supseteq & \mathsf{b}
\end{array}
$$

Final:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\alpha & \supseteq & \mathsf{a} & \alpha & \Longrightarrow & \mathsf{a} \\
\alpha & \supseteq & \mathsf{b} & \alpha & \Longrightarrow & \mathsf{b} \\
\beta & \supseteq & \mathsf{cons}(\gamma, \beta) & \beta & \Longrightarrow & \mathsf{cons}(\gamma, \beta) \\
\beta & \supseteq & \mathsf{nil} & \beta & \Longrightarrow & \mathsf{nil} \\
\gamma & \supseteq & \mathsf{a} & \gamma & \Longrightarrow & \mathsf{a} \\
\gamma & \supseteq & \mathsf{b} & \gamma & \Longrightarrow & \mathsf{b}
\end{array}
$$

Other Operators

- Intersection: consider the following rule

$$\begin{array}{l} \alpha \supseteq \beta \cap \gamma \\ \beta \supseteq c(\beta_1, \beta_2) \;\leadsto\; \alpha \supseteq c(\beta_1 \cap \gamma_1, \beta_2 \cap \gamma_2) \\ \gamma \supseteq c(\gamma_1, \gamma_2) \end{array}$$

  - but this constraint is not in normal form!
  - we must introduce new variables δ_1, δ_2 and write $\alpha \supseteq c(\delta_1, \delta_2)$, $\delta_1 \supseteq \beta_1 \cap \gamma_1$, $\delta_2 \supseteq \beta_2 \cap \gamma_2$
  - new variables $\Rightarrow$ new set expressions
    $\Rightarrow$ termination problems
  - need to carefully control introduction of new variables

- Complement: cannot incrementally construct least solutions
  - very different techniques needed

Implementation

- Issues
  - frequent term comparison $\Rightarrow$ hash consing
  - minimize steps $\Rightarrow$ dependency directed updating
  - intersection $\Rightarrow$ exploit redundancies

- Engineering is important!

- Complexity
  - polynomial time algs for restricted problems
  - exponential for general intersection

Empirical Results

- Dynamic Typing (Henglein)
  - near linear algorithm
  - 1000 lines/sec
- Set-Based Analysis for ML
  - $O(n^3)$ complexity
  - 500 lines/sec
- OO inference for SELF (Agesen, Palsberg, & Schwartzbach)
  - $O(n^3)$ complexity
  - very efficient in practice
- Set-Based Analysis for LP
  - EXPTIME complexity
  - 10-50 lines/sec
- Analysis of FL
  - EXPTIME complexity
  - 10-20 lines/sec

How Do We Obtain Constraints From Programs?

- Use intuition to construct syntax directed rules for constraint generation.
  - prove correctness using subject reduction, induction, abstractions and equivalences, etc.
- Can the constraints be made more accurate?
- What is the underlying model?
- First we consider some examples ...

Imperative Program Example

- Consider an imperative program with data-constructors.
- Introduce a set variable $[\![x]\!]$ for each variable x.

$$\boxed{\begin{array}{lcl} x & := & \mathrm{cons}(1,\ \mathrm{cons}(2,\ \mathrm{nil})) \\ y & := & \mathrm{car}(x) \\ x & := & \mathrm{cdr}(x) \end{array}}$$

$$\begin{array}{lcl} [\![x]\!] & \supseteq & \mathrm{cons}(1,\ \mathrm{cons}(2,\ \mathrm{nil})) \\ [\![y]\!] & \supseteq & \mathrm{cons}_1^{-1}([\![x]\!]) \\ [\![x]\!] & \supseteq & \mathrm{cons}_2^{-1}([\![x]\!]) \end{array}$$

Logic Program Example

- Introduce a set variable $[\![X]\!]$ for each variable X.
- Introduce a set variable γ_p for each predicate p.

```
p(X) ← q(X), r(X).
q(a).
q(b).
r(b).
r(c).
```

$$\begin{aligned}
\gamma_p &\supseteq \mathsf{p}([\![\mathsf{X}]\!]) \\
[\![\mathsf{X}]\!] &\supseteq \mathsf{q}^{-1}(\gamma_q) \cap \mathsf{r}^{-1}(\gamma_r) \\
\gamma_q &\supseteq \mathsf{q}(\mathsf{a}) \cup \mathsf{q}(\mathsf{b}) \\
\gamma_r &\supseteq \mathsf{r}(\mathsf{b}) \cup \mathsf{r}(\mathsf{c})
\end{aligned}$$

A Methodology: Set-Based Analysis

- Set constraints associate *sets* with expressions.
  - Certain "inter-variable" dependencies ignored.
  - An "independent-attribute" approximation.
- *Set-based semantics*: approximate program variables as sets.
- This defines a program's *set-based approximation*.
- Using the set-based semantics, set constraints are constructed to coincide exactly with the program's set-based approximation.
- Identifies canonical set constraints for a program.
  - Correspond to a simple program model.
  - Additional approximation can tune accuracy/efficiency tradeoffs.

Set-Based Program Approximation

- Environments $\Rightarrow$ "Set Environments"

$$\begin{array}{l} [x \mapsto 1,\ y \mapsto 2] \\ [x \mapsto 3,\ y \mapsto 4] \end{array} \quad \leadsto \quad \begin{array}{l} X \mapsto \{1,3\} \\ Y \mapsto \{2,4\} \end{array}$$

- Expressions $\Rightarrow$ Set Expressions
  - (x,y): $\{(1,2),\ (3,4)\}$ $\quad \left\{ \begin{array}{l} (1,2),\ (3,4) \\ (1,4),\ (3,2) \end{array} \right\}$
  - (x,x): $\{(1,1),\ (3,3)\}$ $\quad \left\{ \begin{array}{l} (1,1),\ (3,3) \\ (1,3),\ (3,1) \end{array} \right\}$

Imperative Example Revisited

$$\boxed{\begin{array}{lcl} \mathsf{x} & := & \mathsf{cons}(1,\ \mathsf{cons}(2,\ \mathsf{nil})) \\ \mathsf{y} & := & \mathsf{car}(\mathsf{x}) \\ \mathsf{x} & := & \mathsf{cdr}(\mathsf{x}) \end{array}}$$

$$\begin{array}{lcl} [\![\mathsf{x}]\!] & \supseteq & \mathsf{cons}(1,\ \mathsf{cons}(2,\ \mathsf{nil})) \\ [\![\mathsf{y}]\!] & \supseteq & \mathsf{cons}_1^{-1}([\![\mathsf{x}]\!]) \\ [\![\mathsf{x}]\!] & \supseteq & \mathsf{cons}_2^{-1}([\![\mathsf{x}]\!]) \end{array}$$

$$\begin{array}{lcl} [\![\mathsf{x}]\!]_1 & \supseteq & \mathsf{cons}(1,\ \mathsf{cons}(2,\ \mathsf{nil})) \\ [\![\mathsf{y}]\!]_2 & \supseteq & \mathsf{cons}_1^{-1}([\![\mathsf{x}]\!]_1) \\ [\![\mathsf{x}]\!]_2 & \supseteq & [\![x]\!]_1 \ \cap\ \neg\mathsf{cons}(\top, \top) \\ [\![\mathsf{x}]\!]_3 & \supseteq & \mathsf{cons}_2^{-1}([\![\mathsf{x}]\!]_2) \end{array}$$

Constraint-Based Analysis and Abstract Interpretation

- What advantages do constraint techniques have over standard AI algorithms?

- No distinction between backwards and forwards flow.

- Uniform treatment of infinite height domains.
  - No narrowing or widening needed.

- Constraint solving is incremental.
  - Supports separate analysis/compilation.

Set Constraints Viewed as AI

- Abstract domain for values
  — all sets of values.

- Abstract domain for environments
  — a mapping from variables into sets of values.

- Abstraction of operations
  — set-based version of operation.

Infinite Domain Example

```
X := cons(a, nil);
while X <> nil do
   X := cons(a, X);
```

0. $[\![X]\!] = \{[]\}$
1. $[\![X]\!] = \{[], [a]\}$
2. $[\![X]\!] = \{[], [a], [a,a]\}$
3. $[\![X]\!] = \{[], [a], [a,a], [a,a,a]\}$

and so forth ...

$$[\![X]\!] \supseteq \text{cons}(a, \text{nil})$$
$$[\![X]\!] \supseteq \text{cons}(a, [\![X]\!] \cap \neg \text{nil})$$

$$[\![X]\!] \Rightarrow \text{cons}(a, \text{nil})$$
$$[\![X]\!] \Rightarrow \text{cons}(a, [\![X]\!])$$

Remarks:

- Could insert widening operator around loop.
- But in general this will not yield same result.

Special Case: Analysis of ML

- Using set-based analysis, an analysis for ML can be developed that builds on older algorithms (control flow, grammar methods).

- There is a one-to-one correspondence between program expressions and set expressions during solving of these constraints.

- This constraint solving process can therefore be viewed as a manipulation of sets of program labels.

- Correctness of the set constraint algorithm implies that in this case reasoning about program labels is complete for reasoning about sets.

How Far Do Set Constraints Go?

- Analyses that are in the framework:
  - Remy's record encoding.
  - Henglein's dynamic typing (partially).
  - Tree grammar techniques (Jones & Muchnick)
  - Simple pointer analysis (Hendren's thesis)
  - Effect inference algorithms (monomorphic)
  - 2-level intersection types

- Example constraint analyses that don't fit:
  - Tofte's region analysis (polymorphic recursion).
  - Yellin et. al. data structure analysis.

- Analyses not based on constraints:
  - Graph types (Klarlund & Schwartzbach)
  - Lisp heap analysis (Wang & Hilfinger)
  - Abstract interpretation in higher-order domains.

General Open Problems and Directions

- Treating polymorphism.
  - Well understood in type inference.
  - In program analyses, mostly *ad hoc* except for direct adaptations of type inference.
- Global vs. local analysis.
  - Global: analyze entire program at once.
  - Local: analysis of whole from analysis of parts.
  - When is local analysis possible?
- Analysis of pointer/heap structures.
  - Pointer analysis currently a big topic in imperative community.
  - Constraint techniques for analyzing recursive data structures long-studied in Lisp/FP community.
  - No communication?

Conclusions

- Constraint-based analysis has a useful point of view.
- Large body of useful results:
  - constraint languages
  - resolution algorithms
  - frameworks for justification (e.g., set-based analysis)
- Set constraints are a very useful, but by no means the only, constraint formalism for program analysis.