


### 2.3.4 Run-time implication checking

Predicate implication and branch precedence can be computed statically,  since they do not rely on the actual arguments of a message. Socrates also provides an interface for computing them dynamically at run time (e.g. in a reflective meta-predicate like those constructed by `subtype-of`—see section 4.4):

- `(pred-implies? pred-proc1 pred-proc2)` returns `#t` if `pred-proc1` can be proven to imply `pred-proc2`, or `#f` otherwise.
- `(branch-precedes? branch1 branch2)` returns `#t` if `branch1` has precedence over `branch2`, or `#f` otherwise.
- `(precedent-branches branches)` returns a list of branches in the `branches` list that have the highest precedence—in other words, the minima of the `branches` set according to the partial order defined by `branch-precedes?`.

# Summary of Comments on dissertation- july17.pdf

---

Page: 13


---

Sequence number: 1

Author: KJL

Subject: Note

Date: 7/18/2004 11:56:32 AM

 why? If predicates include recursive predicates, implication is undecidable?

---

## Chapter 3

# Predicate implication

When two plain branches or two around-branches are both applicable to a decision point, precedence between the two branches is determined by the implication relationship between their predicates: if one branch's predicate logically implies the other's, but not vice versa, then the first branch precedes the second. This chapter describes the algorithm used by Socrates to determine whether one predicate implies another. Since a decision point predicate can be any arbitrary Socrates expression, the algorithm can only approximate the implication relation, because it's undecidable in general whether one expression implies another in a Turing-complete language. The algorithm is conservative (assuming no side effects) in that it may report that a predicate does not imply another when in fact it does, but it will never report that a predicate does imply another when it doesn't. This is analogous to static type systems that may reject type-correct programs that it can't prove correct.

### 3.1 The implication algorithm



The predicate implication algorithm in Socrates is conceptually simple: a predicate  $p1$  is considered to logically imply a predicate  $p2$  when the following **implication procedure** is a tautology, i.e., always returns a true value:


```
(lambda/src args (or (not (apply p1 args)) (apply p2 args)))
```

Sequence number: 1

Author: KJL

Subject: Note

Date: 7/18/2004 4:50:59 PM

 Too much detail too soon. Describe why your approximation is practically useful and describe the amolication algorithm first without referring to Scheme.

---

assumptions, returning `#f` if the argument is known to be true and `#t` if the argument is known to be false, and otherwise returning an application expression. A new partial procedure can be associated with a procedure using `(set-pproc! proc pproc)`.

### 3.1.6 Checking the current assumptions

Several of the rules in section 3.1.3 involve checking a partial value against the current assumptions. This may reduce the partial value into a constant true or false value or a simplified expression.

- If the boolean context flag is set and the partial value is a true value, an abstraction expression, or implied by one of the true assumptions (see section 3.1.7, then the partial value is replaced with the value `#t`.
- If the partial value is false, implies one of the false assumptions, or is disjoint from one of the true assumptions (see section 3.1.8, then the partial value is replaced with the value `#f`.

[TBD: distribute-if]

### 3.1.7 Implication between partial values

[TBD]

### 3.1.8 Disjointness of partial values

[TBD]



Sequence number: 1

Author: KJL

Subject: Note

Date: 7/18/2004 5:04:43 PM



We need a good short description why your implication RestrP works.

The problem is: predicate implication GenP is undecidable so we define a restricted form of predicate implication RestrP so that RestrP implies GenP and so that RestrP does not throw out too many good cases.

We have seen this before: Grammar non-ambiguity is undecidable.

We define RestrNonAmbig = LL(1). RestrNonAmbig does not eliminate too many useful grammars: by adding more syntax we can make them LL(1). Etc.

Describe why partial evaluation is the right technology for the given task.

---

## Chapter 5

# Domino puzzle



The domino puzzle project was inspired by a logic puzzle that occasionally appears in Games magazine: use a set of dominos to cover a grid of numbers, where each domino must cover two adjacent numbers that match the domino. Using Socrates and PLT Scheme’s MrEd graphical user interface toolkit [14], I developed a graphical application that randomly generates a (solvable) grid, displays it and a set of dominos, and allows the user to place dominos onto the grid by pointing and clicking.

The first separation of concerns is between the model and the view. The model concern is implemented by a set of classes and methods representing the board and the dominos, while the view concern is implemented by a set of methods that create MrEd widgets corresponding to objects in the model.

### 5.1 The model concern


The top-level class in the model is `polyomino-puzzle`. (I’ve generalized the problem from dominos to polyominos; the main procedure `make-domino-puzzle` just makes a `polyomino-puzzle` that happens to consist of dominos.) A `polyomino-puzzle` consists of a `polyomino-set` and a board. A `polyomino-set` is a set of polyominos. A polyomino is a matrix of tiles. A tile has a label, which can be any value, as well as a rotation, which is an integer between 0 and 3 inclusive, representing the number of 90°

Sequence number: 1

Author: KJL

Subject: Note

Date: 7/18/2004 10:23:33 PM

 We need some kind of quantitative measure? I would like to see statements like:

Program P in language L has "separation of concern quality" (SOCQ)  $x$  and the corresponding  
Socrates program has larger SOCQ. Can you make such statements?

---