Lecture 9

Pete Manolios
Northeastern

Normal Forms

- Minimizing DNF has many applications
  - this is used to analyze the reliability of safety-critical systems
- CNF is the input format of modern SAT solvers
  - this is the so-called DIMACS format
  - modern SAT solvers can solve industrial problems with 1M variables
- There are many other “normal” forms for Boolean formulae
  - decision trees: widely used in machine learning
  - BDDs: very powerful representation used in verification, AI, program analysis, …

Set Theory Connections

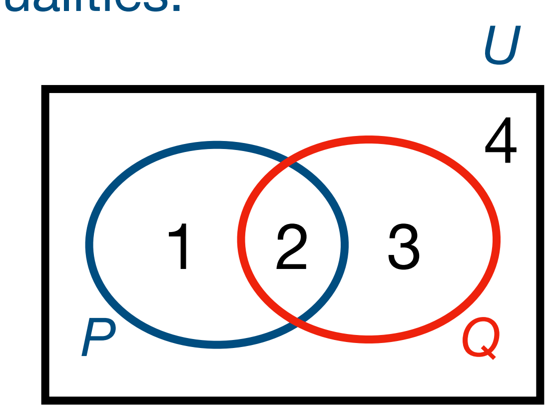
- Set Theory forms the foundations of mathematics
- Set Theory provides foundations of ACL2s:
  - the universe, *U*, is a set
  - recognizers, predicates, etc. in ACL2s are defined in terms of sets
  - atoms in a propositional skeleton are predicates (subsets of *U*)
- It turns out that there are interesting connections between propositional logic and set theory
- Here is an example
  - $p \land (p \lor q) \equiv p$ is valid (in propositional logic) iff
  - $P \cap (P \cup Q) = P$ is valid (in set theory)
  - there are obvious similarities in the two formulas above
- Let's explore the connections a *little* bit

Set Theory Connections

- Boolean Algebra of (non-empty) *X:* a non-empty subset of the 2^X closed under union, intersection and complementation (with respect to *X*)
  - Let *U* be the ACL2s universe
  - Then $B = \{\emptyset, U\}$ is the smallest Boolean algebra of *U*
  - The largest Boolean algebra of *U* is 2^U
- *B* is isomorphic to propositional logic: $\emptyset$ for F and *U* for T
- $\vee, \wedge, \neg$ correspond, respectively, to the (set theoretic) $\cup, \cap, \neg$
- In a Boolean algebra, atoms correspond to unary predicates, *e.g.*, in 2^U:
  - for clarity's sake, we use upper case vars to indicate atoms in 2^U
  - let P be $\{$x $\in U$: (integerp x)$\}$
  - let Q be $\{$x $\in U$: (neg-rationalp x)$\}$
  - so $P \wedge Q$ (in 2^U) means $P \cap Q = \{$x $\in U$: (negp x)$\}$
- A Boolean algebra formula is valid if = U, *e.g.*: $P \vee \neg P$ (in 2^U) means $P \cup \neg P = U$
- In general, a formula in 2^U corresponds to the subset of *U* for which it holds

Set Theory Connections

- Boolean Algebra of (non-empty) *X:* a non-empty subset of the 2^X closed under union, intersection and complementation (with respect to *X*)
- $\vee$, $\wedge$, $\neg$ correspond, respectively, to the (set theoretic) $\cup$, $\cap$, $\neg$
- In general, a formula in 2^U corresponds to the subset of *U* for which it holds
- Can extend Boolean algebra with $\Rightarrow$, $\equiv$, etc, using the propositional equalities:
  - $P \Rightarrow Q$ is $\neg P \vee Q$: $S = \{x \in U :$ (implies ($P\ x$) ($Q\ x$))$\}$
  - $P \equiv Q$ is $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$: $W = \{x \in U :$ (iff ($P\ x$) ($Q\ x$))$\}$
- *The equalities of propositional logic & Boolean algebra are the same!*
  - Propositional logic validity: $p \vee \neg p$
  - Boolean algebra: $P \vee \neg P$ (in 2^U) is valid since $= U$
  - Check the rest of the equalities in the notes
- The result is useful when analyzing propositional logic formulas, *e.g.*:
  - $p \wedge (p \vee q) \equiv p$ is valid iff
  - $P \cap (P \cup Q) = P$ is valid (because $P \equiv Q$ is valid iff $P = Q$ holds)



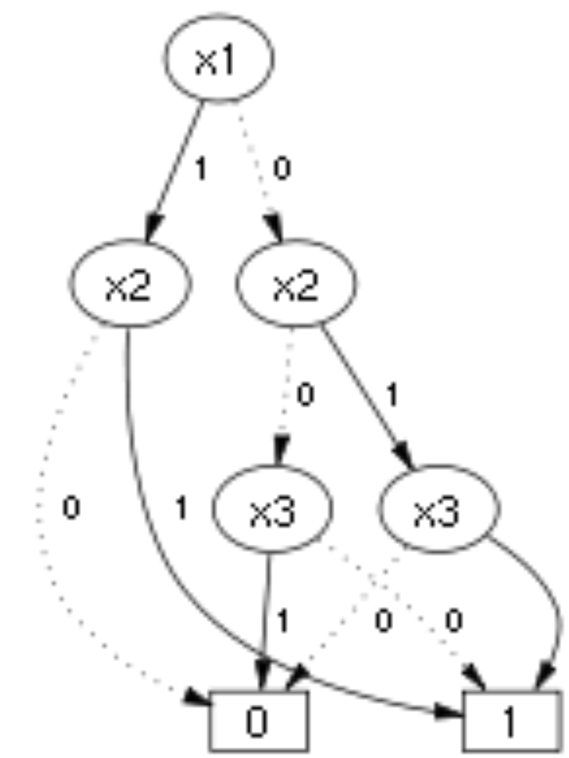
Which regions are in *S*?
2,3,4

Regions for $Q \Rightarrow P$?
2,1,4

Which regions are in *W*?
2,4
(intersection of above)

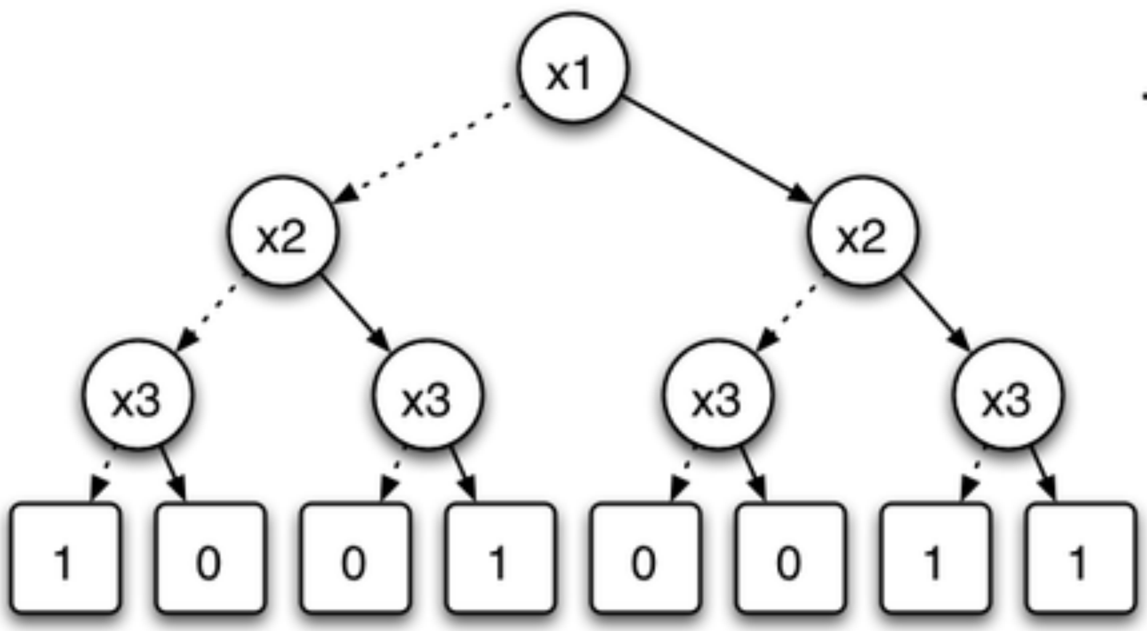
BDDs and Decision Trees

- A BDD on $x_1, \ldots, x_n$ is a DAG $G=(V, E)$ where
  - exactly 1 vertex has indegree 0 (the root)
  - all vertices have outdegree 0 (leaves) or 2 (inner nodes)
  - the inner nodes are labeled from $\{x_1, \ldots, x_n\}$
  - the leaves are labeled from {0, 1}
  - one of the edges from an inner node is labeled by 0; the other by 1
- The BDD $G=(V, E)$ represents a Boolean function, say f
  - for any assignment A in B^n, f(A) is computed recursively from root
  - if we reach a leaf, return the label
  - for inner nodes, say labeled with x_i, take the edge labeled by $A(x_i)$
- A decision tree is a BDD whose graph is a tree
- A BDD is an OBDD if there is a permutation on $p=\{1,2, \ldots, n\}$ s.t. for all edges (u, v) in E, where u, v are labeled by x_i, x_j, we have that $p_i < p_j$
- An OBDD is an ROBDD if it has no isomorphic subgraphs and all children are distinct

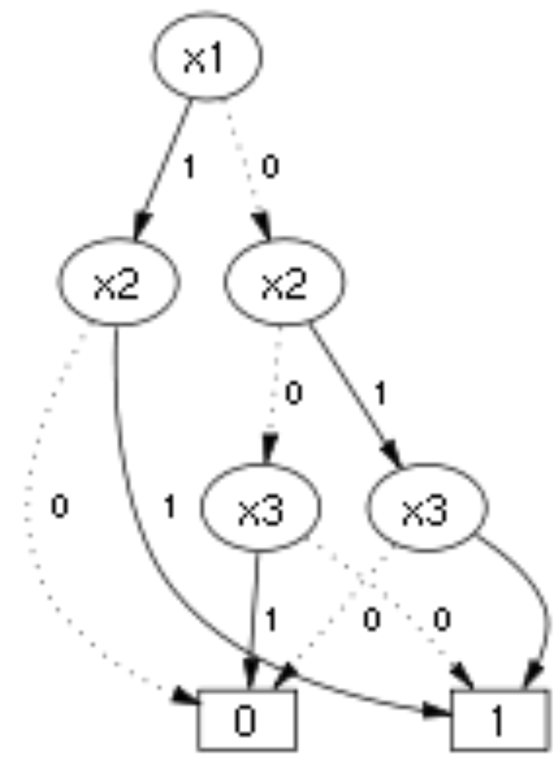


Images from Wikipedia

BDDs and Decision Trees



x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Decision Tree for f

ROBDD for f

How do we generate DNF from a decision tree? ROBDD?

Images from Wikipedia

BDDs

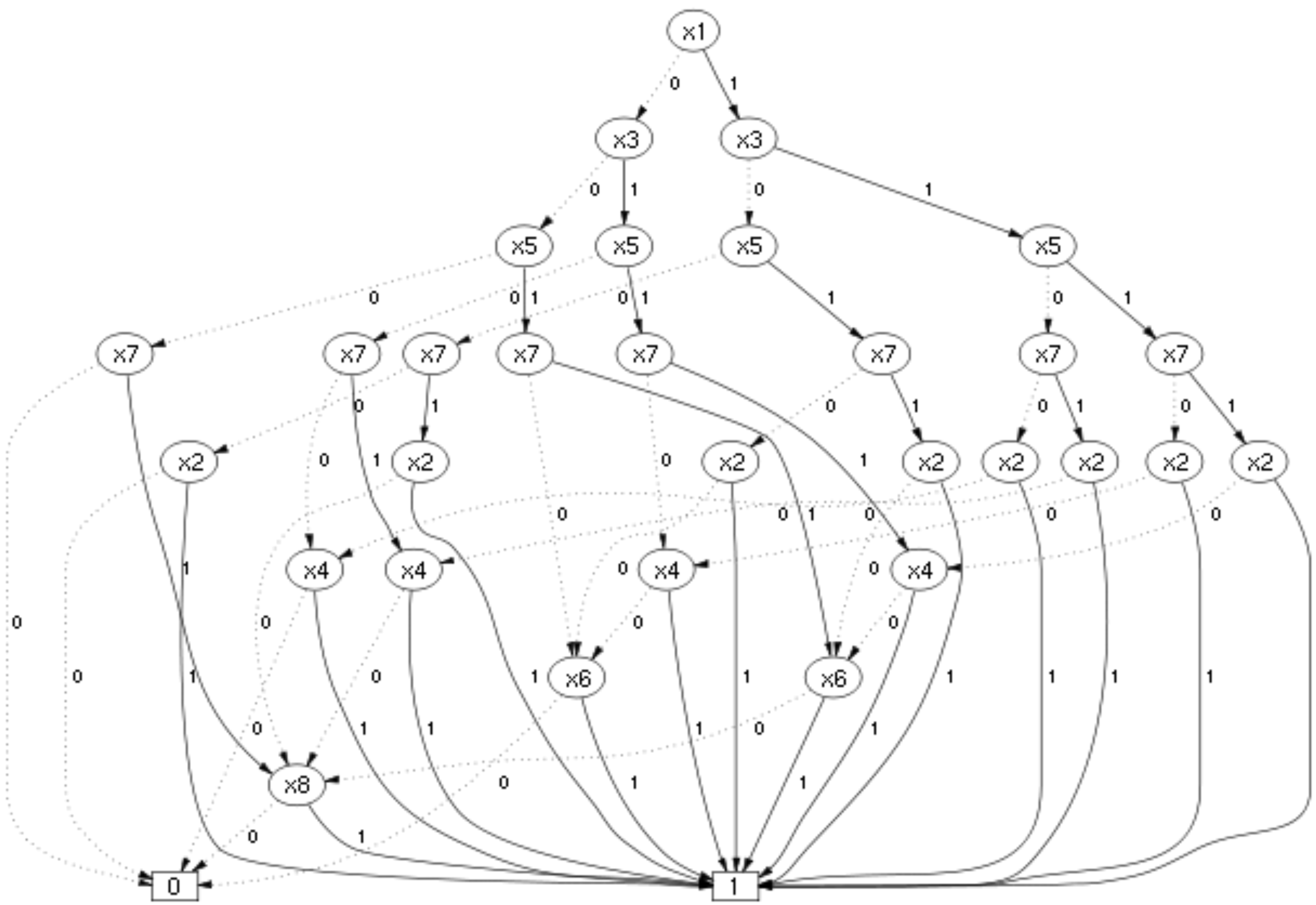
- Decision trees are widely used, e.g., in machine learning (ID3, C4.5, …)
- BDDs are widely used (BDD usually means ROBDD)
  - Popularized by Bryant
  - Very efficient algorithms for constructing, manipulating BDDs
  - Used in verification, synthesis, fault trees, security, AI, model checking, static analysis, …
  - Bryant's paper was the most cited research paper (at some point)
  - Many BDD packages available
- Once a variable ordering is selected, BDDs are canonical!
  - Construct decision tree using Shannon expansion and merge isomorphic nodes, remove nodes who children are equal until you reach a fixpoint
  - To see, this note that BDDs are essentially DFA that recognize strings in $\{0,1\}^n$ and such automata can be minimized (note nodes with equal children remain)
  - So, checking equality is just pointer equality (with appropriate data structures)
  - Can be used for model checking: represent set of reachable states & transition system with BDDs
  - Bryant, Clarke, Emerson & McMillan got 1998 Paris Kanellakis Award for symbolic model checking

BDD Break

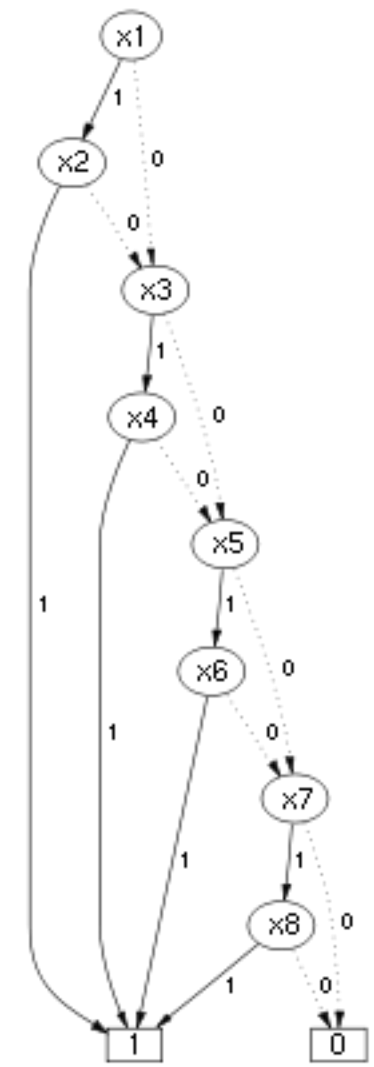
- Made the safety-analysis repo public; see link from slides
- Find someone you haven't spoken to yet
- Come up with an example formula over 4 variables where variable order matters wrt BDD size

Variable Ordering for BDDs

Variable ordering matters: find the best ordering is hard.



Bad Ordering



Good Ordering

What function is this?

Images from Wikipedia

Projects & Presentations

- Talk with me regarding projects
  - Set up 1/2 - 1 hour slots to go over project ideas
- Some ideas (groups 1-2)
  - Better induction proofs
  - Refinement: verification
  - Distributed system verification: perimeter monitoring example, etc
  - CyC: ontology engineering (Doug Lenat)
  - AI & FM: Reasoning about programs
  - Reproduce interesting result
  - Survey paper on some FM topic
  - Harrison’s book in ACL2s

Algorithms for SAT

- Modern SAT solvers accept input in CNF
  - Dimacs format:
  - 1 -3 4 5 0
  - 2 -4 7 0
  - …
- Davis & Putnam Procedure (DP)
  - Dates back to the 50's
  - Based on resolution
  - Helps to explain learning

DP SAT Algorithm

- Davis Putnam (1960)
- Input: CNF formula
- Output: SAT/UNSAT
- Idea: apply three rules until
  - Derive the empty clause: UNSAT (identity of $\vee$ is false)
  - No clauses remain: SAT (identity of $\wedge$ is true)
- Three “rules”
  - Pure literal rule (affirmative-negative rule)
  - Unit resolution rule (unit propagation, BCP, 1-literal rule)
  - Resolution (Called consensus, also used for logic minimization)

Pure Literal Rule

- Given F, a set of clauses, and literal ℓ such
  - ℓ appears in F
  - $\neg\ell$ does not appear in F
  - remove all clauses containing ℓ
- Equisatisfiable because we can make ℓ true
- Notice that this always simplifies F
- Modern SAT solvers tend to not use the rule (efficiency)

Boolean Constraint Propagation

Unit resolution rule:

$$\frac{C, \neg\ell \qquad \ell}{C}$$

- BCP: given a set of clauses including $\{\ell\}$
  - remove all other clauses containing ℓ (subsumption)
  - remove all occurrences of $\neg\ell$ in clauses (unit resolution)
  - repeat until a fixpoint is reached

Resolution

Resolution rule:

$$\frac{C, v \qquad D, \neg v}{C, D} \quad \neg v, v \notin C, D$$

Resolution rule:

$$\frac{C_i, p \qquad D_i, \neg p}{C_i, D_i} \quad \neg p \notin C_i \in P, p \notin D_i \in N$$

- Soundness of rule: above line implies below line
- If below line is SAT, so is above line (w/ side conditions)
- Given literal *p*, set of clauses *S*, let *P* be the clauses in *S* that contain *p* only positively and let *N* be the clauses that contain *p* only negatively. Let *E* be the rest of the clauses. Then *S* is SAT iff *S'* is SAT, where *S'*= *E* U the set of all *p*-resolvents of *P* and *N*.
- Proof: If *A* is an assignment for *S*, then if *A*(*p*)=true, all clauses in *N*, with ¬*p* removed are satisfied, so each *p*-resolvent is satisfied. Similarly if *A*(*p*)=false. If *A* is an assignment for *S*', then it satisfies all *Ci* or all *Di*: suppose it doesn't satisfy *Ck*, then it must satisfy all *Di*. If it satisfies all *Ci*, let *A'*(*p*)=false, else *A'*(*p*)=true and *A'*(*x*)=*A*(*x*) otherwise.

Resolution Example

Resolution rule:

$$\frac{C, v \qquad D, \neg v}{C, D}$$

C, D are clauses, ¬v∉C and v∉D

Given literal *p*, set of clauses *S*, let *P* be the clauses in *S* that contain *p* only positively and let *N* be the clauses that contain *p* only negatively. Let *E* be the rest of the clauses. Then *S* is SAT iff *S'* is SAT, where *S'*= *E* U the set of all *p*-resolvents of *P* and *N*.

$$\{\{\neg p, q, r, s\}, \{p, \neg q, s\}, \{\neg p, \neg q, r, \neg s\}, \boxed{\{p, \neg r, \neg s\}}, \{\neg p, \neg q, \neg r\}, \{p, q\}, \{\neg p, \neg q, s\}\}$$

Resolve on *q*

$$\{\neg p, p, r, s\}$$

$$\{\boxed{\{p, \neg r, \neg s\}}, \{\neg p, r, s\}, \{p, s\}\}$$

Notice that clauses that contain a literal and its negation can be thrown away. Why?

Resolution Example

Resolution rule:

$$\frac{C, v \qquad D, \neg v}{C, D}$$ C, D are clauses, ¬v∉C and v∉D

Given literal *p*, set of clauses *S*, let *P* be the clauses in *S* that contain *p* only positively and let *N* be the clauses that contain *p* only negatively. Let *E* be the rest of the clauses. Then *S* is SAT iff *S'* is SAT, where *S'*= *E* U the set of all *p*-resolvents of *P* and *N*.

$\{\{\neg p, q, r, s\}, \{p, \neg q, s\}, \{\neg p, \neg q, r, \neg s\}, \{p, \neg r, \neg s\}, \{\neg p, \neg q, \neg r\}, \{p, q\}, \{\neg p, \neg q, s\}\}$

Resolve on *q* $\{\neg p, p, r, s\}$

$\{\{p, \neg r, \neg s\}, \{\neg p, r, s\}, \{p, s\}\}$

Notice that clauses that contain a literal and its negation can be thrown away. Why?

Resolve on *r*

$\{\{p, s\}\}$ Sat, resolve on *p* to get {} or use pure literal rule

How do we generate a satisfying assignment? Next homework

DP SAT Algorithm

- Input: CNF formula, Output: SAT/UNSAT
- Base case: empty clause: UNSAT
- Base case: no clauses: SAT
  - Apply these two rules until fixpoint
    - Pure literal rule
    - BCP
  - Choose var, say *x*, perform all possible resolutions, remove trivial clauses and clauses containing *x*
  - Repeat
- Existentially quantify variables, one at a time
- Problem: space blow-up